PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 13/00 | A2

(11) International Publication Number: **WO 98/55932**

(43) International Publication Date: 10 December 1998 (10.12.98)

(21) International Application Number: PCT/US98/10549

(22) International Filing Date: 22 May 1998 (22.05.98)

(30) Priority Data:

08/869,148	4 June 1997 (04.06.97)	US
08/869,277	4 June 1997 (04.06.97)	US

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

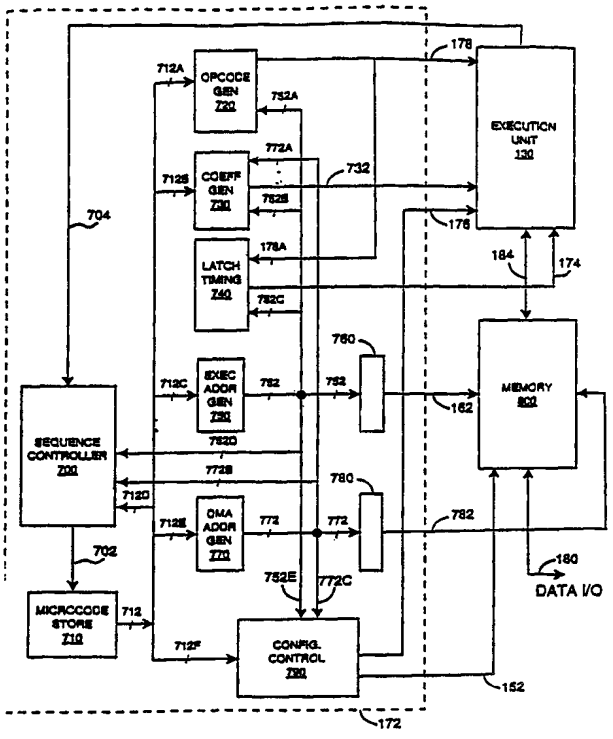
(71)(72) Applicant and Inventor: RUBINSTEIN, Richard [US/US]; 3418 N.E. 83rd Avenue, Vancouver, WA 98662 (US).

(74) Agents: STOLOWITZ, Micah, D. et al.; 1030 S.W. Morrison, Portland, OR 97205 (US).

(54) Title: PROCESSOR INTERFACING TO MEMORY-CENTRIC COMPUTING ENGINE

(57) Abstract

A memory subsystem that is partitioned into two or more blocks of memory space in which one block of the memory communicates with an I/O or DMA channel to load data, while the other block of memory simultaneously communicates with one or more execution units that carry out arithmetic operations on data in the second block. Results are written back to the second block of memory. Upon conclusion of that process, the memory blocks are effectively "swapped" so that the second block, now holding processed (output) data, communicates with the I/O channel to output that data, while the execution unit communicates with the first block, which by then has been filled with new input data. Methods and apparatus are shown for implementing this memory swapping technique in real time so that the execution unit is never idle. A method of interfacing a processor bus to a computation engine having a microprogrammable memory–centric controller and an array of memory, comprising the steps of providing a predetermined series of microcode instructions for execution by the MCC; selecting a start address within the series of microcode instructions for carrying out a corresponding operation; and executing the series of microcode instructions in the MCC beginning at the selected start address so as to carry out the corresponding operation in the engine. The present invention is useful in a wide variety of signal processing applications including programmable MPEG encode and decode, graphics, speech processing, image processing, array processors, etc. In telecommunications, the invention can be used, for example, for switching applications in which multiple I/O channels are operated simultaneously.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

PROCESSOR INTERFACING TO MEMORY-CENTRIC COMPUTING ENGINE

Related application data:

This application claims priority from U.S. application Serial No. 08/869,148, filed June 4, 1997, and co-pending U.S. application Serial No. 08/869,277, also filed June 4, 1997, which was a continuation of U.S. application Serial no. 08/821326 filed March 21, 1997.

FIELD OF THE INVENTION

The present invention is generally in the field of digital computing and, more specifically, is directed to methods and apparatus for interfacing between a processor or bus and an execution subsystem or "engine" that employs shared, reconfigurable memory in a highly flexible, microprogrammable, memory-centric architecture.

BACKGROUND OF THE INVENTION

A. Introduction

The prior application, Serial no. 08/821326, entitled "Shared, Reconfigurable Memory Architectures for Digital Signal Processing" described the need to improve digital signal processing performance while containing or reducing cost. That application describes improved computer architectures that utilize available memory resources more efficiently by providing for shared and reconfigurable memory so as to reduce I/O processor requirements for computation intensive tasks such as digital signal processing. The memory systems described in the prior case are *shared* in the sense that a given block of memory can first be configured for access by the CPU, for example to load data, and then "swapped" so that the same block of physical memory can be directly accessed by an

execution unit, for example a DSP execution unit, to carry out various calculations on that data. After the calculations are completed, the same block of memory can be "swapped" once again, so that the CPU has immediate access to the results.

The memory is *reconfigurable* in a variety of ways, as described below, so as to allocate memory resources as between the CPU and the execution unit (or multiple execution units) in the most efficient manner possible. Reconfiguring the memory can include forming memory blocks of various sizes; selecting write (input) sources; selecting read (destination) targets; selecting word size, and so forth. Various particulars and alternative embodiments are set forth below, so as to enable one skilled in the art to implement shared, reconfigurable memory architectures. The parent case described the invention with reference to digital signal processing. However, DSP is just one example of computation-intensive calculation. The concepts of the prior case as well as the present invention are applicable to a wide variety of execution tasks, including but not limited to DSP and related tasks such as motion picture encoding, decoding, and encryption, decryption, etc.

B. Memory Centric Controller

Another aspect of the prior application is a memory-centric DSP controller ("MDSPC"). The MDSPC was described as providing memory address generation and a variety of other control functions, including reconfiguring the memory as summarized above to support a particular computation in the execution unit. The name "MDSPC" was appropriate for that controller in the context of the parent case, in which the preferred embodiment was described for digital signal processing. However, the principles of the parent cases and the present invention are not so limited. Accordingly, the related application entitled MEMORY CENTRIC CONTROLLER uses its title (or "MCC)" to describe the a controller which is functionally similar to the "MDSPC" introduced in the earlier parent case. The MEMORY CENTRIC CONTROLLER application describes the structure and operation of the MCC in greater detail, and includes detailed

description of interfacing between the reconfigurable memory referred to above and one or more execution units. More specifically, the MCC application describes "bandwidth matching" methods and apparatus to ensure that the execution unit can be operated at maximum throughput even where the memory, e.g. DRAM, is relatively slow.

It is no less important to system performance, however, to ensure sufficient bandwidth at the interface between the memory-centric engine and the host processor bus. Further, bandwidth alone is not enough; compatibility with existing standard interfaces or bus specifications is highly advantageous for reasons explained below. The present application thus is directed to methods and apparatus for interfacing between a processor or bus and a memory-centric execution subsystem or "engine" of the type described in the two related applications identified above.

C. DMA and Custom CPU Interface

The memory-centric architecture as described in the prior applications interfaces with a microprocessor core using essentially two types of interface, each of which is known in prior art for memory access. The first is DMA — Direct Memory Access. A DMA transfer allows moving a block of data into or from a memory without requiring cycles of the CPU processor or core. Instead, a DMA controller, generally hardware, handles the transfer and notifies the CPU when the transfer is done. The DMA is given a start location or address and a length count, and provides addressing to the memory. In the prior applications, we noted that the MDSPC (or MCC) includes DMA hardware for handling such tasks. We also refer to an I/O channel as one that allows DMA transfers. The DMA controller can be on board (or within) a processor or part of another component of the system. Thus one method for interfacing is to treat the MCC as a coprocessor to the core and communicate with it (i.e. transfer data) by employing I/O or DMA methodologies analogous to those known in prior art. One problem with DMA transfer, however, is the DRAM in the memory-centric engine may not provide

adequate transfer rates to support the host bus. Alternatively, one can interface with the MCC by utilizing an existing co-processor interface provided on the particular processor core implementation being deployed.

A more direct interface is to modify the processor core architecture to accommodate the memory-centric engine directly. This solution is preferred in terms of optimizing performance and minimum transistor count. It provides maximum computing engine performance together with optimum core processor performance. The problem with that approach is that custom modification of an existing core architecture or hardware implementation is a difficult and time consuming -- and therefore expensive -- task. Core designs are complex and modifying them requires substantial engineering resources and time-to-market. Therefore, custom modification of each core processor design as a prerequisite to implementation of a memory-centric engine can be expected to impede implementation of the MCC engine.

In view of the foregoing background, the need remains for an architecture solution that will provide a standard interface, in other words i.e., take advantage of known and existing interfacing methods and apparatus. In other words, the need remains for a way to interface to a memory-centric computing engine, that provides enhanced performance and high bandwidth without requiring custom modification of the host processor core, yet still provide performance improvements over standard interfaces such as DMA and standard I/O channels or interfacing via a co-processor bus.

SUMMARY OF THE INVENTION

The present invention is directed to improved hardware architectures for digital signal processing, and more specifically, is directed to "memory-centric" methods and apparatus for improved performance in digital signal processing or "DSP" systems. As mentioned above, improvements in DSP performance have been achieved by providing special arithmetic units or "execution units" that are optimized to carry out the arithmetic operations that are commonly required in

DSP -- mainly multiplication and addition -- at very high speed. One example of such an execution unit is the "DAU" (data execution unit) provided in the WE DSP32C chip from AT&T. The AT&T execution unit, and others like it, provide relatively fast, floating point arithmetic operations to support computation-intensive applications such as speech, graphics and image processing.

While many improvements have been made in floating-point execution units, pipelined architectures, decreased cycle times, etc., known DSP systems generally work with standard memory systems. For example, DRAM integrated circuits are used for reading input data and, on the output side, for storing output data. DSP data is moved into and out of the DRAM memory systems using known techniques such as multiple-ported memory, DMA hardware, buffers, and the like. While such systems benefit from improvements in memory speed and density, data transfer remains a relative bottleneck. I have reconsidered these known techniques and discovered that significant gains in performance and flexibility can be achieved by focusing on the memory, in addition to the execution unit, and by providing improvements in methods and circuits for moving data efficiently among data sources (such as a host processor bus or I/O channel), memory subsystems, and execution units. Since the focus is on the memory, I coined the term "memory-centric" computing.

One aspect of the invention is a memory subsystem that is partitioned into two or more blocks of memory space. One block of the memory communicates with an I/O or DMA channel to load data, while the other block of memory simultaneously communicates with one or more execution units that carry out arithmetic operations on data in the second block. Results are written back to the second block of memory. Upon conclusion of that process, the memory blocks are effectively "swapped" so that the second block, now holding processed (output) data, communicates with the I/O channel to output that data, while the execution unit communicates with the first block, which by then has been filled with new input data. Methods and apparatus are shown for implementing this memory swapping technique in real time so that the execution unit is never idle.

Another aspect of the invention provides for interfacing two or more address generators to the same block of memory, so that memory block swapping can be accomplished without the use of larger multi-ported memory cells.

The present invention is useful in a wide variety of signal processing applications including programmable MPEG encode and decode, graphics, speech processing, image processing, array processors, etc. In telecommunications, the invention can be used, for example, for switching applications in which multiple I/O channels are operated simultaneously.

A further aspect of the invention provides for partitioning the memory space into two or more memory "segments" -- with the ability to selectively assign one or more such segments to form a required block of memory. Thus, for example, one block of memory can be configured to include say, four segments, and be associated with an execution unit, while another block of memory is configured to include only one segment and may be assigned to an I/O or DMA channel. This flexibility is useful in matching the memory block size to the requirements of an associated execution unit for a particular operation, say a recursive type of digital filter such as an Infinite Impulse Response (IIR) filter. Memory segments can be of arbitrary size as will be shown in more detail later.

Importantly, the memory is readily "reconfigurable" so that it can adapt to the particular calculations required. Several implementations are disclosed herein. In one embodiment, the memory reconfiguration is controlled by configuration control signals. The configuration control signals may be generated based upon "configuration bits" which can be downloaded from a core processor, instruction decoder, or durable memory, for reconfiguring the memory responsive to the task at hand. In another arrangement, the configuration bits are stored in extended bit positions in the regular memory, so that pointers or traps can be used in software to reconfigure the hardware. A further aspect of the invention provides for generating configuration control signals in an improved address generator or in a novel Memory-centric DSP Controller ("MDSPC"). The new address generation techniques include both reconfiguring and addressing the memory to support a

particular computation in the execution unit.

Another feature of the invention is that memory blocks can be reconfigured both in depth, i.e. number or words or rows, as well as in width (word size). This flexibility simplifies, and speeds, memory I/O for various applications, and provides great flexibility in a single DSP system, which can be implemented as a separate "co-processor" or "on-board" with a general purpose or other core processor. For example, the memory word size can be easily configured to match that of the I/O channel currently in use. The invention can be implemented in both von Neumann as well as Harvard architectures.

A further aspect of the invention, again directed to improvements in data flow, provides ways to interface multiple blocks of memory, in combination with one or more execution units. In some applications, parallel execution units can be used to advantage. Another aspect of the invention is a system that is readily configurable to take advantage of the available execution resources. The configuration bits described above also can include controls for directing data to and from multiple execution units as and when appropriate.

The invention further anticipates an execution unit that can be reconfigured in several ways, including selectable depth (number of pipeline stages) and width (i.e. multiple word sizes concurrently). Preferably the pipelined execution unit(s) includes internal register files with feedback. The execution unit configuration and operation also can be controlled by execution unit configuration control signals. The execution unit configuration control signals can be determined by "configuration bits" stored in the memory, or stored in a separate "configuration table". The configuration table can be downloaded by the host core processor, and/or updated under software control. Preferably, the configuration control signals are generated by the MDSPC controller mentioned above executing microcode. This combination of reconfigurable memory, together with reconfigurable execution units, and the associated techniques for efficiently moving data between them, provides an architecture that is highly flexible. Microcoded software can be used to take advantage of this architecture so as to

achieve new levels of performance in DSP systems. Because the circuits described herein require only one-port or two-port memory cells, they allow higher density and the associated advantages of lowered power dissipation, reduced capacitance, etc. in the preferred integrated circuit embodiments, whether implemented as a stand-alone coprocessor, or together with a standard processor core, or by way of modification of an existing processor core design. An important feature of the architectures described herein is that they provide a tightly coupled relationship between memory and execution units. This feature provides the advantages of reducing internal interconnect requirements, thereby lowering power consumption. In addition, the invention provides for doing useful work on virtually all clock cycles. This feature minimizes power consumption as well.

A memory-centric computing engine provides any one of several standard interfaces. For example, a standard memory interface can be provided. In this case, the interface includes address lines, data lines, and control signals such as RAS/ (row address strobe), CAS/ (column address strobe), write enable, output enable, etc. In other words, the MC engine presents itself to the processor like a memory. However, techniques are shown below that provide SRAM speed at the interface combined with DRAM density in the engine so as to accommodate complex computations requiring substantial amounts of data.

This invention provides simplified yet high performance interaction between the engine and the host. For example, the processor can load a calculation "problem" (data) into the apparent "memory" and, after execution, simply read out the results using the same methods as a standard DRAM memory read operation. In the meantime, the MC engine performed the necessary operations.

The interface is configurable under microcode control. It can accommodate a memory interface, CPU bus interface, or indeed virtually any given standard or specification. Examples include the PCI Local Bus, VME Ebus, RAMBUS, etc. Other presently known bus interface standards are the Sun SBUS, PCMCIA, Multibus and the ISA and EISA -- commonly called the IBM AT bus. Detailed technical specifications of these standards are widely published and therefore need

not be detailed here.

According to one aspect of the invention, a method of interfacing a processor bus to a computation engine having a microprogrammable memory-centric controller and an array of memory is defined. The claimed method includes the steps of providing a predetermined series of microcode instructions for execution by the MCC; selecting a start address within the series of microcode instructions for carrying out a corresponding operation; and executing the series of microcode instructions in the MCC beginning at the selected start address so as to carry out the corresponding operation in the engine. The series of microcode instructions can be stored in a non-volatile memory accessible to the MCC; or in a non-volatile external memory accessible to the MCC. Alternatively, using the disclosed architecture, microcode can be downloaded under processor control to a separate microcode storage memory accessible to the MCC, or into the array of memory (DRAM) in the computation engine.

Another aspect of the invention also directed to interfacing with a bus is a method of downloading the microcode instructions by first asserting a predetermined address; decoding the predetermined address in the MCC; and in response to the decoding step, configuring the engine for storing the microcode instructions under processor control into the array of memory. The decoding can be done by "hard-wired" logic or it can be microcode programmable, or a combination of the two.

According to another aspect of the invention, the computing engine includes an SRAM buffer memory and the memory array comprises an array of DRAM memory. In operation, a write operation for example includes storing data from the external bus into the SRAM buffer memory and then transferring the stored data from the buffer memory into the DRAM array.

According to a further aspect of the invention, moving the stored data to the DRAM array includes writing a series of data words into a latch and then writing the contents of the latch into the DRAM array in a single write operation so as to improve matching access time of the SRAM buffer memory to access time of the

DRAM array.

Thus the present invention provides methods and apparatus to reconfigure hardware, move data, etc. control executions, conduct testing, and provide high-speed interface, while maintaining compatibility with known standard bus interface specifications. Moreover, because easily reconfigured under software control, can easily be adapted and changed to a different interface as may be required. For example, the data format, word size, error correction bits, addressing format etc. can all be changed within the bounds of the available number of "wires" of signal lines. Thus, the memory-centric engine can interface with, say the PCI bus in one application, while the same or an identical chip can comply with RAMBUS standards in another application. Bandwidth of course is key, and we show below the methods and apparatus for interfacing at the appropriate bus speed, using SRAM buffer cells and memory block swapping techniques.

The foregoing and other objects, features and advantages of the invention will become more readily apparent from the following detailed description of a preferred embodiment of the invention which proceeds with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a system level block diagram of an architecture for digital signal processing (DSP) using shared memory according to the present invention.

FIG. 2 illustrates circuitry for selectively coupling two or more address generators to a single block of memory.

FIG. 3 is a block diagram illustrating portions of the memory circuitry and address generators of Fig. 1 in a fixed-partition memory configuration.

FIG. 4 shows more detail of address and bit line connections in a two-port memory system of the type described.

FIGS. 5A-5C illustrate selected address and control signals in a Processor

Implementation of a DSP system, *i.e.* a complete DSP system integrated on a single chip.

FIG. 6A illustrates an alternative embodiment in which a separate DSP program counter is provided for accessing the memory.

FIG. 6B illustrates an alternative embodiment in which an MDSPC accesses the memory.

FIGS. 7A-B are block diagrams that illustrate embodiments of the invention in a Harvard architecture.

FIG. 8 is a conceptual diagram that illustrates a shared, reconfigurable memory architecture according to the present invention.

FIG. 9 illustrates connection of address lines to a shared, reconfigurable memory with selectable (granular) partitioning of the reconfigurable portion of the memory.

FIG. 10 illustrates a system that implements a reconfigurable segment of memory under bit selection table control.

FIG. 11A is a block diagram illustrating an example of using single-ported RAM in a DSP computing system according to the present invention.

FIG. 11B is a table illustrating a pipelined timing sequence for addressing and accessing the one-port memory so as to implement a "virtual two-port" memory.

FIG. 12 illustrates a block of memory having at least one reconfigurable segment with selectable write and read data paths.

FIG. 13A is a schematic diagram showing detail of one example of the write selection circuitry of the reconfigurable memory of Fig. 12.

FIG. 13B illustrates transistor pairs arranged for propagating or isolating bit lines as an alternative to transistors 466 in Fig. 13A or as an alternative to the bit select transistors 462, 464 of Fig. 13A.

FIG. 14 is a block diagram illustrating extension of the shared, reconfigurable memory architecture to multiple segments of memory.

FIG. 15 is a simplified block diagram illustrating multiple reconfigurable

memory segments with multiple sets of sense amps.

FIGS. 16A-16D are simplified block diagrams illustrating various examples of memory segment configurations to form memory blocks of selectable size.

FIG. 17 is a block diagram of a DSP architecture illustrating a multiple memory block to multiple execution unit interface scheme in which configuration is controlled via specialized address generators.

FIGS. 18A-18C are simplified block diagrams illustrating various configurations of segments of a memory block into association with multiple execution units.

FIG. 19 is a simplified block diagram illustrating a shared, reconfigurable memory system utilizing common sense amps.

FIG. 20 is a simplified block diagram illustrating a shared, reconfigurable memory system utilizing multiple sense amps for each memory segment.

FIG. 21 is a timing diagram illustrating memory swapping cycles.

FIG. 22A is a block diagram illustrating memory swapping under bit table control.

FIG. 22B is a block diagram illustrating memory swapping under MDSPC control.

FIG. 23A is a block diagram of an embodiment of the MCC having a general architecture.

FIG. 23B is a block diagram of an embodiment of the MCC having an architecture adapted to operate in cooperation with a core processor.

FIG. 23C is a block diagram of an embodiment of the MCC having an architecture adapted to increase data throughput.

FIGS. 24A and 24B are examples of a memory mapped microcode for the embodiments of FIGS. 23A-C.

FIG. 25 is a functional block diagram of an example of a configuration of the execution unit of FIGS. 23A-C.

FIG. 26 is a functional block diagram of another example of a

configuration of the execution unit of FIGS. 23A-C.

FIG. 27 is a functional block diagram of an example of a memory configuration and execution unit interface for a high bandwidth memory to execution unit interface according to the present invention.

FIG. 28 is a functional block diagram of an embodiment of the selection logic and wide input latch of FIG. 27.

FIG. 29 is a functional block diagram of an embodiment of the high bandwidth memory to execution unit interface of FIG. 27 which illustrates a multi-stage pipelined execution unit.

FIG. 30A is a waveform diagram of an example of the clock signals for the circuit of FIG. 27.

FIG. 30B is a functional block diagram of an example of the high speed timing and control block of FIG. 27.

FIG. 31 is a functional block diagram illustrating an example of a configuration for the reconfigurable multi-stage pipelined execution unit of FIG. 27 which utilizes a multi-ported register file to obtain feedback in a pipelined data path.

FIG. 32 is a functional block diagram illustrating another example of a configuration for the reconfigurable multi-stage pipelined execution unit of FIG. 27 which demonstrates a different feedback path in a pipelined data path.

FIG. 33 is a functional block diagram illustrating an embodiment of the circuit of FIGS. 23A-C which includes feedback signal paths to support fast branching within an execution unit.

FIG. 34 is a process flow diagram illustrating branching in the circuit of FIG. 33.

FIG. 35 is a process flow diagram illustrating branching in the circuit of FIG. 33 and FIG. 23C for a more complex example involving speculative execution.

FIG. 36 is a diagram of the memory configuration for the process illustrated in FIG. 35.

Fig. 37 is a simplified block diagram showing interconnection of a digital processor and memory subsystem to a memory-centric computing engine via a standard bus interface.

Fig. 38 is a block diagram showing more detail of the memory and control portions the memory-centric computing engine 16 of Fig. 1.

Fig. 39 is a simplified block diagram showing interconnection of the memory-centric computing engine to a flash or other memory for accessing stored microcode.

Fig. 40 is a simplified block diagram illustrating one implementation of the memory-centric computing engine that includes a bus interface controller with buffer memory.

Fig. 41 is a block diagram of the memory-centric computing engine of Fig. 4 showing greater detail of the interface controller and including multiple blocks of interface buffer memory.

Fig. 42 is a conceptual diagram of a DRAM array in which the first or leading edge bits of each row of the array are implemented with SRAM cells to provide high speed access.

Fig. 43 is a block diagram of a pseudo-static RAM architecture utilizing a DRAM array together with a bus interface controller.

Fig. 44 is a block diagram illustrating interfacing SRAM buffer to DRAM array for bandwidth matching.

Fig. 45 is a simplified diagram showing addressing for accessing the microcode memory by the external bus or by the MCC.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

FIGURE 1

Fig. 1 is a system-level block diagram of an architecture for memory and computing-intensive applications such as digital signal processing. In Fig. 1, a

microprocessor interface 40 includes a DMA port 42 for moving data into a memory via path 46 and reading data from the memory via path 44. Alternatively, a single, bi-directional port could be used. The microprocessor interface 40 generically represents an interface to any type of controller or microprocessor. The interface partition indicated by the dashed line 45 in Fig. 1 may be a physical partition, where the microprocessor is in a separate integrated circuit, or it can merely indicate a functional partition in an implementation in which all of the memory and circuitry represented in the diagram of Fig. 1 is implemented on board a single integrated circuit. Other types of partitioning, use of hybrid circuits, etc., can be used. The microprocessor interface (DMA 42) also includes control signals indicated at 52. The microprocessor or controller can also provide microcode (not shown) for memory control and address generation, as well as control signals for configuration and operation of the functional execution units, as described later.

Because the present invention may be integrated into an existing processor or controller core design, so that both the core processor and the present invention reside in the same integrated circuit, reference will be made herein to the core processor meaning the processor that the present invention has been attached to or integrated with.

In Fig. 1, a two-port memory comprises the first memory block 50, labeled "A" and a second memory block 60, labeled "B." The memory is addressed by a source address generator 70 and a destination address generator 80. A functional execution unit 90 also is coupled to the two-port memory, left and right I/O channels, as illustrated at block B. Preferably, these are not conventional two-port memory I/O ports; rather, they have novel structures described later.

In operation, the interface 44, 46 to the two-port memory block A is a DMA interface that is in communication with the host processor or controller 40. Block A receives data coefficients and optionally other parameters from the controller, and also returns completed data to the controller that results from various DSP, graphics, MPEG encode/decode or other operations carried out in

the execution unit 90. This output data can include, for example, FFT results, or convolution data, or graphics rendering data, etc. Thus the single memory can alternately act as both a graphics frame buffer and a graphics computation buffer memory.

Concurrently, the memory block "B" (60) interfaces with the functional execution unit 90. The functional execution unit 90 receives data from the two-port memory block B and executes on it, and then returns results ("writeback") to the same two-port memory structure. The source address generator 70 supplies source or input data to the functional execution unit while the destination address generator 80 supplies addresses for writing results (or intermediate data) from the execution unit to the memory. Put another way, source address generator 70 provides addressing while the functional execution unit is reading input data from memory block B, and the destination address generator 80 provides addressing to the same memory block B while the functional execution unit 90 is writing results into the memory.

As mentioned above, when the execution unit has completed its work on the data in block B, the memory effectively "swaps" blocks A and B, so that block B is in communication with the DMA channel 42 to read out the results of the execution. Conversely, and simultaneously, the execution unit proceeds to execute on the new input data in block A. This "swapping" of memory blocks includes several aspects, the first of which is switching the memory address generator lines so as to couple them to the appropriate physical block of memory.

In an alternative embodiment, the system can be configured so that the entire memory space (blocks A and B in the illustration) are accessed first by an I/O channel, and then the entire memory swapped to be accessed by the processor or execution unit. In general, any or all of the memory can be reconfigured as described. The memory can be SRAM, DRAM or any other type of random access semiconductor memory or functionally equivalent technology. DRAM refresh is provided by address generators, or may not be required where the speed of execution and updating the memory (access frequency) is sufficient to obviate

refresh.

FIGURE 2

Figure 2 illustrates one way of addressing a memory block with two (or more) address generators. Here, one address generator is labeled "DMA" and the other "ADDR GEN" although they are functionally similar. As shown in Fig. 2, one of the address generators 102 has a series of output lines, corresponding to memory word lines. Each output line is coupled to a corresponding buffer (or word line driver or the like), 130 to 140. Each driver has an enable input coupled to a common enable line 142. The other address generator 104 similarly has a series of output lines coupled to respective drivers 150 to 160. The number of word lines is at least equal to the number of rows of the memory block 200. The second set of drivers also have enable inputs coupled to the common enable control line 142, but note the inverter "bubbles" on drivers 130 to 140, indicating the active-low enables of drivers 150 to 160. Accordingly, when the control line 142 is low, the DMA address generator 102 is coupled to the memory 200 row address inputs. When the control line 142 is high, the ADDR GEN 104 is coupled to the memory 200 row address inputs. In this way, the address inputs are "swapped" under control of a single bit. Alternative circuitry can be used to achieve the equivalent effect. For example, the devices illustrated can be tri-state output devices, or open collector or open drain structures can be used where appropriate. Other alternatives include transmission gates or simple pass transistors for coupling the selected address generator outputs to the memory address lines. The same strategy can be extended to more than two address sources, as will be apparent to those skilled in the art in view of this disclosure.

FIGURE 3

Figure 3 is a block diagram illustrating a physical design of portions of the memory circuitry and address generators of Fig. 1 in a fixed-partition configuration. By "fixed partition" I mean that the size of memory block A and the size of memory block B cannot change dynamically. In Fig. 3, the memory block

A (50) and block B (60) correspond to the same memory blocks of Fig. 1. The memory itself preferably is dynamic RAM, although static RAM or other solid state memory technologies could be used as well. In memory block B, just two bits or memory cells 62 AND 64 are shown by way of illustration. In a typical implementation, the memory block is likely to include thousands or even millions of rows, each row (or word) being perhaps 64 or more bits wide. A typical memory block using today's technology is likely to be one or two megabytes. The memory blocks need not be of equal size. Neither memory depth nor word size is critical to the invention.

Two bits are sufficient here to illustrate the concept without unduly complicating the drawing. The source address generator 70 is coupled to both memory blocks A and B. In block B, the top row includes a series of cells including bit cell 62. In fact, the source address generator preferably has output lines coupled to all of the rows of not only block B, but block A as well, although only one row line is illustrated in block A. Note also that corresponding address lines from the AG 70 and the DMA 102 are shown as connected in common, e.g. at line 69. However, in practice, these address lines are selectable as described above with reference to Fig. 2.

A destination address generator 80 similarly is coupled to the row lines of both blocks of memory. Memory cells 62 and 64 are full two-ported cells on the same column in this example. Thus, either source AG 70 or DMA 102 address the left port, while either destination AG 80 or DMA 100 address the right port. A write select multiplexer 106 directs data either from the DMA (42 in Fig. 1) (or another block of memory) or from the execution unit 90, responsive to a control signal 108. The control signal is provided by the controller or microprocessor of Fig, 1, by a configuration bit, or by an MDSPC. The selected write data is provided to column amplifiers 110, 112 which in turn are connected to corresponding memory cell bit lines. 110 and 112 are bit and /bit ("bit bar") drivers. Below cell 64 is a one-bit sense amplifier 116. A bit output from the sense amp 116 is directed, for example, to a latch 72. Both the DMA and the

execution unit are coupled to receive data from latch 72, depending on appropriate control, enable and clock signals (not shown here). Or, both the DMA and the execution path may have separate latches, the specifics being a matter of design choice. Only one sense amp is shown for illustration, while in practice there will be at least one sense amp for each column. Use of multiple sense amps is described later.

FIGURE 4

Fig. 4 shows more detail of the connection of cells of the memory to source and destination address lines. This drawing shows how the source address lines (when asserted) couple the write bit line and its complement, i.e. input lines 110,112 respectively, to the memory cells. The destination address lines couple the cell outputs to the read bit lines 114, 115 and thence to sense amp 116. Although only one column is shown, in practice write and read bit lines are provided for each column across the full width of the memory word. The address lines extend across the full row as is conventional.

FIGURES 21, 22A AND 22B

Timing

Fig. 21 is a conceptual diagram illustrating an example for the timing of operation of the architecture illustrated in Fig. 1. T0A, T1A, etc., are specific instances of two operating time cycles T0 and T1. The cycle length can be predetermined, or can be a parameter downloaded to the address generators. T0 and T1 are not necessarily the same length and are defined as alternating and mutually exclusive, i.e. a first cycle T1 starts at the end of T0, and a second cycle T0 starts at the end of the first period T1, and so on. Both T0 and T1 are generally longer than the basic clock or memory cycle time.

Fig. 22A is a block diagram of a single port architecture which will be used to illustrate an example of functional memory swapping in the present invention during repeating T0 and T1 cycles. Execution address generator 70

addresses memory block A (50) during T0 cycles. This is indicated by the left (T0) portion of AG 70. During T1 cycles, execution address generator 70 addresses memory block B (60), as indicated by the right portion of 70. During T1, AG 70 also receives setup or configuration data in preparation for again addressing Mem Block A during the next T0 cycle. Similarly, during T0, AG 70 also receives configuration data in preparation for again addressing Mem Block B during the next T1 cycle.

DMA address generator 102 addresses memory block B (60) during T0 cycles. This is indicated by the left (T0) portion of DMA AG 102. During T1 cycles, DMA address generator 102 addresses memory block A (50), as indicated by the right portion of 102. During T1, DMA AG 102 also receives setup or configuration data in preparation for again addressing Mem Block B during the next T0 cycle. Similarly, during T0, DMA 102 also receives configuration data in preparation for again addressing Mem Block A during the next T1 cycle.

During a T0 cycle, the functional execution unit (90 in Fig. 1) is operating continuously on data in memory block A 50 under control of execution address generator 70. Simultaneously, DMA address generator 102 is streaming data into memory block B 60.

At the beginning of a T1 cycle, memory blocks A and B effectively swap such that execution unit 90 will process the data in memory block B 60 under control of execution address generator 70 and data will stream into memory block A 50 under control of DMA address generator 102. Conversely, at the beginning of a T0 cycle, memory blocks A and B again effectively swap such that execution unit 90 will process the data in memory block A 50 under control of execution address generator 70 and data will stream into memory block B 60 under control of DMA address generator 102.

In Fig. 22B, the functions of the execution address generator and DMA address generator are performed by the MDSPC 172 under microcode control.

FIGURES 5A-C

Processor Implementation

The preferred architecture for implementation in a processor application, as distinguished from a coprocessor application, is illustrated in Figs. 5A-C. In Fig. 5A, a two-port memory again comprises a block A (150) and a block B (160). Memory block B is coupled to a DSP execution unit 130. An address generator 170 is coupled to memory block B 160 via address lines 162. In operation, as before, the address generator unit is executing during a first cycle T0 and during time T0 is loading parameters for subsequent execution in cycle T1. The lower memory block A is accessed via core processor data address register 142A or core processor instruction address register 142B. Thus, in this illustration, the data memory and the instructional program memory are located in the same physical memory. A microprocessor system of the Harvard architecture has separate physical memory for data and instructions. The present invention can be used to advantage in the Harvard architecture environment as well, as described below with reference to Figs. 7A and 7B.

Bit Configuration Tables

Fig. 5A also includes a bit configuration table 140. The bit configuration table can receive and store information from the memory 150 or from the core processor, via bus 180, or from an instruction fetched via the core processor instruction address register 142B. Information is stored in the bit configuration table during cycle T0 for controlling execution during the next subsequent cycle T1. The bit configuration table can be loaded by a series of operations, reading information from the memory block A via bus 180 into the bit configuration tables. This information includes address generation parameters and opcodes. Examples of some of the address parameters are starting address, modulo-address counting, and the length of timing cycles T0 and T1. Examples of op codes for controlling the execution unit are the multiply and accumulate operations necessary for to perform an FFT.

Essentially, the bit configuration table is used to generate configuration

control signal 152 which determines the position of virtual boundary 136 and, therefore, the configuration of memory blocks A and B. It also provides the configuration information necessary for operation of the address generator 170 and the DSP execution unit 130 during the T1 execution cycle time. Path 174 illustrates the execution unit/memory interface control signals from the bit configuration table 140 to the DSP execution unit 130. Path 176 illustrates the configuration control signal to the execution unit to reconfigure the execution unit. Path 178 illustrates the op codes sent to execution unit 130 which cause execution unit to perform the operations necessary to process data. Path 188 shows configuration information loaded from the configuration tables into the address generator 170.

The architecture illustrated in Fig. 5A preferably would utilize the extended instructions of a given processor architecture to allow the address register from the instruction memory to create the information flow into the bit configuration table. In other words, special instructions or extended instructions in the controller or microprocessor architecture can be used to enable this mechanism to operate as described above. Such an implementation would provide tight coupling to the microprocessor architecture.

Memory-centric DSP Controller

Fig. 5B illustrates an embodiment of the present invention wherein the functions of address generator 170 and bit configuration table 140 of Fig. 5A are performed by memory-centric DSP controller (MDSPC) 172. In the embodiment shown in Fig. 5B, the core processor writes microcode for MDSPC 172 along with address parameters into memory block B 150. Then, under core processor control, the microcode and address parameters are downloaded into local memory within MDSPC 172.

A DSP process initiated in MDSPC 172 then generates the appropriate memory configuration control signals 152 and execution unit configuration control signals 176 based upon the downloaded microcode to control the position

of virtual boundary 136 and structure execution unit 130 to optimize performance for the process corresponding to the microcode. As the DSP process executes, MDSPC 172 generates addresses for memory block B 160 and controls the execution unit/memory interface to load operands from memory into the execution unit 130 which are then processed by execution unit 130 responsive to op codes 178 sent from MDSPC 172 to execution unit 130. In addition, virtual boundary 136 may be adjusted responsive to microcode during process execution in order to dynamicly optimize the memory and execution unit configurations.

In addition, the MDSPC 172 supplies the timing and control for the interfaces between memory and the execution unit. Further, algorithm coefficients to the execution unit may be supplied directly from the MDSPC. The use of microcode in the MDSPC results in execution of the DSP process that is more efficient than the frequent downloading of bit configuration tables and address parameters associated with the architecture of Fig. 5A. The microcoded method represented by the MDSPC results in fewer bits to transfer from the core processor to memory for the DSP process and less frequent updates of this information from the core processor. Thus, the core processor bandwidth is conserved along with the amount of bits required to store the control information.

Fig. 5C illustrates an embodiment of the present invention wherein the reconfigurability of memory in the present invention is used to allocate an additional segment of memory, memory block C 190, which permits MDSPC 172 to execute microcode and process address parameters out of memory block C 190 rather than local memory. This saves the time required for the core processor controlled download of microcode and address parameters to local memory in MDSPC 172 that takes place in the embodiment of Fig. 5B. This embodiment requires an additional set of address 192 and data 194 lines to provide the interface between memory block C 190 and MDSPC 172 and address bus control circuitry 144 under control of MDSPC 172 to disable the appropriate address bits from core processor register file 142. This configuration permits simultaneous access of MDSPC 172 to memory block C 190, DSP execution unit 130 to memory block B

and the core processor to memory block A.

Similar to the embodiments shown in Figs. 5A and 5B, virtual boundaries 136A and 136B are dynamically reconfigurable to optimize the memory configuration for the DSP process executing in MDSPC 172.

The bit tables and microcode discussed above may alternatively reside in durable store, such as ROM or flash memory. The durable store may be part of memory block A or may reside outside of memory block A wherein the content of durable store is transferred to memory block A or to the address generators or MDSPC during system initialization.

Furthermore, the DSP process may be triggered by either decoding a preselected bit pattern corresponding to a DSP function into an address in memory block A containing the bit tables or microcode required for execution of the DSP function. Yet another approach to triggering the DSP process is to place the bit tables or microcode for the DSP function at a particular location in memory block A and the DSP process is triggered by the execution of a jump instruction to that particular location. For instance, at system initialization, the microcode to perform a DSP function, such as a Fast Fourier Transform (FFT) or IIR, is loaded beginning at a specific memory location within memory block A. Thereafter, execution of a jump instruction to that specific memory location causes execution to continue at that location thus spawning the DSP process.

FIGURES 6A and 6B

Referring now to Fig. 6A, in an alternative embodiment, a separate program counter 190 is provided for DSP operations. The core controller or processor (not shown) loads information into the program counter 190 for the DSP operation and then that program counter in turn addresses the memory block 150 to start the process for the DSP. Information required by the DSP operations would be stored in memory. Alternatively, any register of the core processor, such as data address register 142A or instruction address register 142B, can be used for addressing memory 150. Bit Configuration Table 140, in addition to generating

memory configuration signal 152, produces address enable signal 156 to control address bus control circuitry 144 in order to select the address register which accesses memory block A and also to selectively enable or disable address lines of the registers to match the memory configuration (i.e. depending on the position of virtual boundary 136, address bits are enabled if the bit is needed to access all of memory block A and disabled if block A is smaller than the memory space accessed with the address bit).

Thus, Fig. 6A shows the DSP program counter 190 being loaded by the processor with an address to move into memory block A. In that case, the other address sources in register file 142 are disabled, at least with respect to addressing memory 150. In short, three different alternative mechanisms are illustrated for accessing the memory 150 in order to fetch the bit configurations and other parameters 140. The selection of which addressing mechanism is most advantageous may depend upon the particular processor architecture with which the present invention is implemented.

Fig. 6B shows an embodiment wherein MDSPC 172 is used to generate addresses for memory block A in place of DSP PC 190. Address enable signal 156 selects between the address lines of MDSPC 172 and those of register file 142 in response to the microcode executed by MDSPC 172. As discussed above, if the microcode for MDSPC 172 resides in memory block A or a portion thereof, MDSPC 172 will be executing out of memory block A and therefore requires access to the content of memory block A.

Memory Arrangement

Referring again to Fig. 5, memory blocks A (150) and B (160) are separated by "virtual boundary" 136. In other words, block A and block B are portions of a single, common memory, in a preferred embodiment. The location of the "virtual boundary" is defined by the configuration control signal generated responsive to the bit configuration table parameters. In this regard, the memory is reconfigurable under software control. Although this memory has a variable

boundary, the memory preferably is part of the processor memory, it is not contemplated as a separate memory distinct from the processor architecture. In other words, in the processor application illustrated by Figs. 5 and 6, the memory as shown and described is essentially reconfigurable directly into the microprocessor itself. In such a preferred embodiment, the memory block B, 160, duly configured, executes into the DSP execution unit as shown in Fig. 5.

In regard to Fig. 5B, virtual boundary 136 is controlled based on the microcode downloaded to MDSPC 172. Similarly, in Fig. 5C, microcode determines the position of both virtual boundary 136A and 136B to create memory block C 190.

FIGURES 7A and 7B

Fig. 7A illustrates an alternative embodiment, corresponding to Fig. 5A, of the present invention in a Harvard-type architecture, comprising a data memory block A 206 and block B 204, and a separate core processor instruction memory 200. The instruction memory 200 in addressed by a program counter 202. Instructions fetched from the instruction memory 200 pass via path 220 to a DSP instruction decoder 222. The instruction decoder in turn provides addresses for DSP operations, table configurations, etc., to an address register 230. Address register 230 in turn addresses the data memory block A 206. Data from the memory passes via path 240 to load the bit configuration tables etc. 242 which in turn configure the address generator for addressing the data memory block B during the next execution cycle of the DSP execution unit 250. Fig. 6 thus illustrates an alternative approach to accessing the data memory A to fetch bit configuration data. A special instruction is fetched from the instruction memory that includes an opcode field that indicates a DSP operation, or more specifically, a DSP configuration operation, and includes address information for fetching the appropriate configuration for the subroutine.

In the embodiment of Fig. 7B, corresponding to the embodiments in Figs. 5B and 5C, MDSPC 246 replaces AG 244 and Bit Configuration Table 242.

Instructions in core processor instruction memory 200 that correspond to functions to be executed by DSP Execution Unit 250 are replaced with a preselected bit pattern which is not recognized as a valid instruction by the core processor. DSP Instruction Decode 222 decodes the preselected bit patterns and generates an address for DSP operations and address parameters stored in data memory A and also generates a DSP control signal which triggers the DSP process in MDSPC 246. DSP Instruction Decode 222 can also be structured to be responsive to output data from data memory A 206 into producing the addresses latched in address register 230.

The DSP Instruction Decode 222 may be reduced or eliminated if the DSP process is initiated by an instruction causing a jump to the bit table or microcode in memory block A pertaining to the execution of the DSP process.

To summarize, the present invention includes an architecture that features shared, reconfigurable memory for efficient operation of one or more processors together with one or more functional execution units such as DSP execution units. Fig. 6A shows an implementation of a sequence of operations, much like a subroutine, in which a core controller or processor loads address information into a DSP program counter, in order to fetch parameter information from the memory. Fig. 6B shows an implementation wherein the DSP function is executed under the control of an MDSPC under microcode control. In Figs. 5A-C, the invention is illustrated as integrated with a von Neumann microprocessor architecture. Figs. 7A and 7B illustrate applications of the present invention in the context of a Harvard-type architecture. The system of Fig. 1 illustrates an alternative stand-alone or coprocessor implementation. Next is a description of how to implement a shared, reconfigurable memory system.

Reconfigurable Memory Architecture

FIGURE 8

Fig. 8 is a conceptual diagram illustrating a reconfigurable memory architecture for DSP according to another aspect of the present invention. In Fig.

8, a memory or a block of memory includes rows from 0 through Z. A first portion of the memory 266, addresses 0 to X, is associated, for example, with an execution unit (not shown). A second (hatched) portion of the memory 280 extends from addresses from X+1 to Y. Finally, a third portion of the memory 262, extending from addresses Y+1 to Z, is associated, for example, with a DMA or I/O channel. By the term "associated" here we mean a given memory segment can be accessed directly by the designated DMA or execution unit as further explained herein. The second segment 280 is reconfigurable in that it can be switched so as to form a part of the execution segment 266 or become part of the DMA segment 262 as required.

The large vertical arrows in Fig. 8 indicate that the execution portion and the DMA portion of the memory space can be "swapped" as explained previously. The reconfigurable segment 280 swaps together with whichever segment it is coupled to at the time. In this block of memory, each memory word or row includes data and/or coefficients, as indicated on the right side of the figure.

Additional "configuration control bits" are shown to the left of dashed line 267. This extended portion of the memory can be used for storing a bit configuration table that provides configuration control bits as described previously with reference to the bit configuration table 140 of Figs. 5A and 6A. These selection bits can include write enable, read enable, and other control information. So, for example, when the execution segment 266 is swapped to provide access by the DMA channel, configuration control bits in 266 can be used to couple the DMA channel to the I/O port of segment 266 for data transfer. In this way, a memory access or software trap can be used to reconfigure the system without delay.

The configuration control bits shown in Fig. 8 are one method of effecting memory reconfiguration that relates to the use of a separate address generator and bit configuration table as shown in Figs. 5A and 7A. This approach effectively drives an address configuration state machine and requires considerable overhead processing to maintain the configuration control bits in a consistent and current

state.

When the MDSPC of Figs. 5B, 5C and 7B is used, the configuration control bits are unnecessary because the MDSPC modifies the configuration of memory algorithmically based upon the microcode executed by the MDSPC. Therefore, the MDSPC maintains the configuration of the memory internally rather than as part of the reconfigured memory words themselves.

FIGURE 9

Fig. 9 illustrates connection of address and data lines to a memory of the type described in Fig. 8. Referring to Fig. 9, a DMA or I/O channel address port 102 provides sufficient address lines for accessing both the rows of the DMA block of memory 262, indicated as bus 270, as well as the reconfigurable portion of the memory 280, via additional address lines indicated as bus 272. When the block 280 is configured as a part of the DMA portion of the memory, the DMA memory effectively occupies the memory space indicated by the brace 290 and the address lines 272 are controlled by the DMA channel 102. Fig. 9 also shows an address generator 104 that addresses the execution block of memory 266 via bus 284. Address generator 104 also provides additional address lines for controlling the reconfigurable block 280 via bus 272. Thus, when the entire reconfigurable segment 280 is joined with the execution block 266, the execution block of memory has a total size indicated by brace 294, while the DMA portion is reduced to the size of block 262.

The address lines that control the reconfigurable portion of the memory are switched between the DMA address source 102 and address generator 104 via switching means 296. Illustrative switching means for addressing a single block of memory from multiple address generators was described above, for example with reference to Fig. 2. The particular arrangement depends in part on whether the memory is single-ported (see Fig. 2) or multi-ported (see Figs. 3-4). Finally, Fig. 9 indicates data access ports 110 and 120. The upper data port 110 is associated with the DMA block of memory, which, as described, is of selectable

size. Similarly, port 120 accesses the execution portion of the memory. Circuitry for selection of input (write) data sources and output (read) data destinations for a block of memory was described earlier. Alternative structures and implementation of multiple reconfigurable memory segments are described below.

It should be noted that the entire block need not be switched *in toto* to one memory block or the other. Rather, the reconfigurable block preferably is partitionable so that a selected portion (or all) of the block can be switched to join the upper or lower block. The granularity of this selection (indicated by the dashed lines in 280) is a matter of design choice, at a cost of additional hardware, e.g. sense amps, as the granularity increases, as further explained later.

FIGURE 10

Fig. 10 illustrates a system that implements a reconfigurable segment of memory 280 under bit selection table control. In Fig. 10, a reconfigurable memory segment 280 receives a source address from either the AG or DMA source address generator 274 and it receives a destination address from either the AG or DMA destination address generator 281. Write control logic 270, for example a word wide multiplexer, selects write input data from either the DMA channel or the execution unit according to a control signal 272. The source address generator 274 includes bit table control circuitry 276. The configuration control circuitry 276, either driven by a bit table or under microcode control, generates the write select signal 272. The configuration control circuitry also determines which source and destination addresses lines are coupled to the memory -- either "AG" (address generator) when the block 280 is configured as part of the an "AG" memory block for access by the execution unit, or the "DMA" address lines when the block 280 is configured as part of the DMA or I/O channel memory block. Finally, the configuration control logic provides enable and/or clock controls to the execution unit 282 and to the DMA channel 284 for controlling which destination receives read data from the memory output data output port 290.

FIGURE 11

Fig. 11 is a partial block/partial schematic diagram illustrating the use of a single ported RAM in a DSP computing system according to the present invention. In Fig. 11, a single-ported RAM 300 includes a column of memory cells 302, 304, etc. Only a few cells of the array are shown for clarity. A source address generator 310 and destination address generator 312 are arranged for addressing the memory 300. More specifically, the address generators are arranged to assert a selected one address line at a time to a logic high state. The term "address generator" in this context is not limited to a conventional DSP address generator. It could be implemented in various ways, including a microprocessor core, microcontroller, programmable sequencer, etc. Address generation can be provided by a micro-coded machine. Other implementations that provide DSP type of addressing are deemed equivalents. However, known address generators do not provide control and configuration functions such as those illustrated in Fig. 10 -- configuration bits 330. For each row of the memory 300, the corresponding address lines from the source and destination blocks 310, 312, are logically "ORed" together, as illustrated by OR gate 316, with reference to the top row of the memory comprising memory cell 302. Only one row address line is asserted at a given time. For writing to the memory, a multiplexer 320 selects data either from the DMA or from the execution unit, according to a control signal 322 responsive to the configuration bits in the source address generator 310. The selected data is applied through drivers 326 to the corresponding column of the memory array 300 (only one column, i.e. one pair of bit lines, is shown in the drawing). For each column, the bit lines also are coupled to a sense amplifier 324, which in turn provides output or write data to the execution unit 326 and to the DMA 328 via path 325. The execution unit 326 is enabled by an execution enable control signal responsive to the configuration bits 330 in the destination address block 312. Configuration bits 330 also provide a DMA control enable signal at 332.

The key here is to eliminate the need for a two-ported RAM cell by using a logical OR of the last addresses from the destination and source registers (located in the corresponding destination or source address generators). Source and destination operations are not simultaneous, but operation is still fast. A source write cycle followed by a destination read cycle would take only a total time of two memory cycles.

FIGURE 12

Fig. 12. The techniques and circuits described above for reconfigurable memory can be extended to multiple blocks of memory so as to form a highly flexible architecture for digital signal processing. Fig. 12 illustrates a first segment of memory 400 and a second memory segment 460. In the first segment 400, only a few rows and a few cells are shown for purposes of illustration. One row of the memory begins at cell 402, a second row of the memory begins at cell 404, etc. Only a single bit line pair, 410, is shown for illustration. At the top of the figure, a first write select circuit such as a multiplexer 406 is provided for selecting a source of write input data. For example, one input to the select circuit 406 may be coupled to a DMA channel or memory block M1. A second input to the MUX 406 may be coupled to an execution unit or another memory block M2. In this discussion, we use the designations M1, M2, etc., to refer generically, not only to other blocks of memory, but to execution units or other functional parts of a DSP system in general. The multiplexer 406 couples a selected input source to the bit lines in the memory segment 400. The select circuit couples all, say 64 or 128 bit lines, for example, into the memory. Preferably, the select circuit provides the same number of bits as the word size.

The bit lines, for example bit line pair 410, extend through the memory array segment to a second write select circuit 420. This circuit selects the input source to the second memory segment 460. If the select circuit 420 selects the bit lines from memory segment 400, the result is that memory segment 400 and the second memory segment 460 are effectively coupled together to form a single

block of memory. Alternatively, the second select circuit 420 can select write data via path 422 from an alternative input source. A source select circuit 426, for example a similar multiplexer circuit, can be used to select this input from various other sources, indicated as M2 and M1. When the alternative input source is coupled to the second memory segment 460 via path 422, memory segment 460 is effectively isolated from the first memory segment 400. In this case, the bit lines of memory segment 400 are directed via path 430 to sense amps 440 for reading data out of the memory segment 400. When the bitlines of memory segment 400 are coupled to the second segment 460, sense amps 440 can be sent to a disable or low power standby state, since they need not be used.

FIGURE 13

Fig. 13 shows detail of the input selection logic for interfacing multiple memory segments. In Fig. 13, the first memory segment bit line pair 410 is coupled to the next memory segment 460, or conversely isolated from it, under control of pass devices 466. When devices 466 are turned off, read data from the first memory segment 406 is nonetheless available via lines 430 to the sense amps 440. The input select logic 426 includes a first pair of pass transistors 426 for connecting bit lines from source M1 to bit line drivers 470. A second pair of pass transistors 464 controllably couples an alternative input source M2 bit lines to drivers 470. The pass devices 462, 464, and 466, are all controllable by control bits originating, for example, in the address generator circuitry described above with reference to Fig. 9. Pass transistors, transmission gates or the like can be considered equivalents for selecting input (write data) sources.

FIGURE 14

Fig. 14 is a high-level block diagram illustrating extension of the architectures of Figs. 12 and 13 to a plurality of memory segments. Details of the selection logic and sense amps is omitted from this drawing for clarity. In general, this drawing illustrates how any available input source can be directed to any

segment of the memory under control of the configuration bits.

Fig. 15 is another block diagram illustrating a plurality of configurable memory segments with selectable input sources, as in Fig. 14. In this arrangement, multiple sense amps 482, 484, 486, are coupled to a common data output latch 480. When multiple memory segments are configured together so as to form a single block, fewer than all of the sense amps will be used. For example, if memory segment 0 and memory segment 1 are configured as a single block, sense amp 484 provides read bits from that combined block, and sense amp 482 can be idle.

Figs. 16A through 16D are block diagrams illustrating various configurations of multiple, reconfigurable blocks of memory. As before, the designations M1, M2, M3, etc., refer generically to other blocks of memory, execution units, I/O channels, etc. In Fig. 16A, four segments of memory are coupled together to form a single, large block associated with input source M1. In this case, a single sense amp 500 can be used to read data from this common block of memory (to a destination associated with M1). In Fig. 16B, the first block of memory is associated with resource M1, and its output is provided through sense amp 502. The other three blocks of memory, designated M2, are configured together to form a single block of memory -- three segments long -- associated with resource M2. In this configuration, sense amp 508 provides output from the common block (3xM2), while sense amps 504 and 506 can be idle. Figs. 16C and 16D provide additional examples that are self explanatory in view of the foregoing description. This illustration is not intended to imply that all memory segments are of equal size. To the contrary, they can have various sizes as explained elsewhere herein.

Fig. 17 is a high-level block diagram illustrating a DSP system according to the present invention in which multiple memory blocks are interfaced to multiple execution units so as to optimize performance of the system by reconfiguring it as necessary to execute a given task. In Fig. 17, a first block of

memory M1 provides read data via path 530 to a first execution unit ("EXEC A") and via path 532 to a second execution unit (EXEC B"). Execution unit A outputs results via path 534 which in turn is provided both to a first multiplexer or select circuit MUX-1 and to a second select circuit MUX-2. MUX-1 provides select write data into memory M1.

Similarly, a second segment of memory M2 provides read data via path 542 to execution unit A and via path 540 to execution unit B. Output data or results from execution unit B are provided via path 544 to both MUX-1 and to MUX-2. MUX-2 provides selected write data into the memory block M2. In this way, data can be read from either memory block into either execution unit, and results can be written from either execution unit into either memory block.

A first source address generator S1 provides source addressing to memory block M1. Source address generator S1 also includes a selection table for determining read/write configurations. Thus, S1 provides control bit "Select A" to MUX--1 in order to select execution unit A as the input source for a write operation to memory M1. S1 also provides a "Select A" control bit to MUX-2 in order to select execution unit A as the data source for writing into memory M2.

A destination address generator D1 provides destination addressing to memory block M1. D1 also includes selection tables which provide a "Read 1" control signal to execution A and a second "Read 1" control signal to execution unit B. By asserting a selected one of these control signals, the selection bits in D1 directs a selected one of the execution units to read data from memory M1.

A second source address generator S2 provides source addressing to memory segment M2. Address generator S2 also provides a control bit "select B" to MUX-1 via path 550 and to MUX-2 via path 552. These signals cause the corresponding multiplexer to select execution unit B as the input source for write back data into the corresponding memory block. A second destination address generator D2 provides destination addressing to memory block M2 via path 560. Address generator D2 also provides control bits for configuring this system. D2 provides a read to signal to execution unit A via path 562 and a read to signal to

execution unit B via path 564 for selectively causing the corresponding execution unit to read data from memory block M2.

Fig. 18A illustrates at a high level the parallelism of memory and execution units that becomes available utilizing the reconfigurable architecture described herein. In Fig. 18A, a memory block, comprising for example 1,000 rows, may have, say, 256 bits and therefore 256 outputs from respective sense amplifiers, although the word size is not critical. 64 bits may be input to each of four parallel execution units E1 - E4. The memory block thus is configured into four segments, each segment associated with a respective one of the execution units, as illustrated in Fig. 18B. As suggested in the figure, these memory segments need not be of equal size. Fig. 18C shows a further segmentation, and reconfiguration, so that a portion of segment M2 is joined with segment M1 so as to form a block of memory associated with execution unit E1. A portion of memory segment M3, designated "M3/2" is joined together with the remainder of segment M2, designated "M2/2", to form a memory block associated with execution unit E2, and so on.

Note, however, that the choice of one half block increments for the illustration above is arbitrary. Segmentation of the memory may be designed to permit reconfigurability down to the granularity of words or bits if necessary.

FIG. 19.

The use of multiple sense amps for memory segment configuration was described previously with reference to Figs. 15 and 16. Fig. 19 illustrates an alternative embodiment in which the read bit lines from multiple memory segments, for example read bit lines 604, are directed to a multiplexer circuit 606, or its equivalent, which in turn has an output coupled to shared or common set of sense amps 610. Sense amps 610 in turn provide output to a data output latch 612, I/O bus or the like. The multiplexer or selection circuitry 604 is responsive to control signals (not shown) which select which memory segment output is

"tapped" to the sense amps. This architecture reduces the number of sense amps in exchange for the addition of selection circuitry 606.

Fig. 20. is a block diagram illustrating a memory system of multiple configurable memory segments having multiple sense amps for each segment. This alternative can be used to improve speed of "swapping" read data paths and reduce interconnect overhead in some applications.

FIGURES 23A-C Memory Centric Controller

Figures 23A-C illustrate an embodiment of MDSPC 172 according to the present invention. Though the MDSPC 172 is described in the context of a DSP controller, it can be configured to perform a wide variety of functions other than DSP that benefit from the use of a flexible data path control architecture, such as data processing engines for compression/ decompression or packet header processing. Therefore, whereas the related application referenced above discussed an MDSPC, the present application will hereinafter utilize the terminology of a Memory-Centric Controller (MCC) to reflect the broader applicability of the present invention.

MCC 172 contains a sequence controller 700 which sequences through a microcode routine stored in microcode store 710. Sequence controller 700 functions in a manner similar to conventional sequencers which are found in a wide variety of applications and which sequence through a set of control instructions in order to control a variety of functional units external to the controller and redirect program flow responsive to microcode instructions and feedback from the external functional units.

However, sequence controller 700 differs from conventional sequencers in its use of a microcode control word for memory centric applications. Memory centric control words include DMA-I/O addresses, execution addresses, opcodes for execution units, reconfiguration and configuration control bits for memory configuration, and latch timing and control for memory interfaces and pipeline control between the memory and execution units.

Microcode store

Microcode store 710 may take a number of forms and be populated in various ways depending upon the context within which the MCC will operate. For instance, microcode store 710 can include read-only memory (ROM), such as EEPROM, flash memory and other forms of non-volatile storage, which contains all or part of the microcode routines to be executed by MCC 172. Alternatively, microcode store 710 may include volatile memory, such as DRAM or SRAM, which is initialized at power-up by an initialization routine contained within a bootstrap memory also included within microcode store 710, that downloads microcode routines from non-volatile storage. These examples would typically be present in an embedded application where the MCC 172 is, at least in part, an independently functioning processor which executes a relatively narrow set of functions.

In the context where MCC 172 is teamed with a core processor, the microcode can be downloaded under control of the core processor. This requires that microcode store 710 be accessible to the core processor, in addition to the sequence controller 700, but provides for greater flexibility and broader range in the functions performed by MCC 172 because a microcode program is downloaded for the specific task required by the core processor, which allows the code to be optimized for the particular function, rather than being adapted for more general functionality, and also allows the use of customized routines developed by a user of the MCC 172.

Alternatively, MCC 172 can configure a portion of memory 800 to contain the microcode instructions and eliminate the need for a separate microcode store 710. However, the overhead involved in downloading the microcode from the core processor to microcode store 710 for each task is likely to be small relative to the time required to process a block of data. The overhead for download, in turn, may be reduced by structuring the microcode to have subroutines shared by a number of higher level functions so that only the microcode specific to the higher

level function need be downloaded for a particular task. In this case, the microcode will include a library of routines and subroutines shared by the higher level functions.

An additional advantage of microcode in MCC 172 is that a complex built-in self test (BIST) can be implemented. A microcode BIST can reconfigure memory 800 and the internal circuitry of MCC 172 to perform extensive testing of the circuitry before a device fabricated with MCC 172 is packaged or shipped. This represents a significant production cost savings. Further, the MCC 172 is able to perform the tests without the need for an extensive on the packaged device, therefore bad devices are selected at wafer sort and final packaged devices will have higher yield thus reducing final device cost. Finally, the BIST microcode does not require additional circuitry nor does it require storage space in the microcode store 710 since it can be replaced with operational microcode programming after the BIST has been performed.

Execution initialization

Initialization of microcode execution in the MCC 172 can also be accomplished in a variety of ways which are generally understood in the conventional art.

One way to initiate execution is through memory mapped function wherein a predetermined address, present upon an address bus from the core processor and corresponding to a specific function, is decoded within the MCC 172 to initiate execution of the microcode for the specific function. FIGS. 24A and 24B illustrate an example of memory mapped function wherein an address of 0x710 corresponds to a Radix 2 FFT function, 0x905 corresponds to a FIR filter, and address 0x2100 corresponds to a convolution function. In this configuration, sequence controller 700 includes decode circuitry which will decode a reserved address value of the core processor address signal and translate the reserved address value into a microcode address in microcode store 710, which corresponds to the function represented by the reserved address value, where microcode

execution is initiated.

The importance of memory mapped function is that MCC operations can be implemented transparently to a core processor having a memory mapped software architecture. No change is necessary to the instructions of the core processor. Integration of the MCC can be as simple as modifying I/O software drivers for each functions in the core processor software to access the addresses corresponding to the function in the MCC. Instead of process flow continuing in the core processor when a memory mapped MCC function is addressed, MCC 172 decodes the address and begins process execution.

Another alternative for initiating execution in MCC 172 includes the use of a memory location or mailbox in memory 800 which is reserved for a status word which is accessed by both the core processor and the MCC 172 in order to communicate processing state information. This approach can be implemented in the configuration illustrated in FIG. 23B. For instance, the MCC 172 will write a predetermined status word to the reserved location to indicate to the core processor that processing is complete before the MCC 172 swaps memory blocks, such as memory blocks A (150) and B (160) in FIG. 23B, within memory 800. The reserved location can exist in memory blocks A and B in which case the core processor will access the reserved location through external bus interface mechanism 142 which supports access by the core processor directly or through another external bus, such as a RAMBUS. The core processor periodically reads the status word in the reserved location to determine when the MCC 172 has swapped the memory so that the core processor can proceed to act upon the data present in the memory block (i.e. transfer the result data from the memory block to an output port, transfer new data for processing into the memory block, etc.). Note that the core processor can move the data itself or, if a DMA-I/O address generator is included, MCC 172 can perform a DMA transfer function under microcode control. Note that a dedicated status register that is separate from memory 800 can be utilized instead of the reserved location.

Alternatively, writing the status word to the reserved location or mailbox

can also be designed to generate an interrupt to the core processor which reads the status word to determine the appropriate interrupt handling routine corresponding to the information in the status word. Implementing an interrupt driven interface between the MCC and the core processor can be useful to integrating the MCC into a memory mapped software architecture in the core processor.

The mailbox or status word architecture can also be used to create communications links between multiple MCC based devices. When one MCC device, a DSP engine for example, completes processing of a block of data, it can write to a status word to a mailbox corresponding to another MCC based device, a high speed data I/O device or FFT engine for example. Writing to the mailbox can generate an interrupt in the other MCC which reads the status word from the mailbox and initiate a function appropriate to the information in the status word. In this manner, considerable intelligence can be distributed in MCC based devices within a system and the interfaces between the MCC based subsystems can be optimized to take advantage of the high throughput capacity of the present architecture without the requirement of core processor resources.

Yet another alternative for execution initialization is to include a command register within the MCC 172, and accessible by the core processor, which receives macro-instructions from the core processor which indicate the DSP functions (or other complex operations) to be performed by the MCC 172 as well as a DMA-I/O function to be performed, along with the corresponding data addresses for each function. For DMA transfers, the macro-instruction command identifies the direction of data movement and includes such parameters as the starting and ending addresses of a block of data in memory 800 to be transferred. The macro-instructions for DSP functions and DMA transfers can be supported by underlying microcode which executes the function indicated by the macro-instruction. Still other alternatives for initiating execution in the MCC 172 include standard handshaking or interrupt driven protocols wherein various microcode programs in microcode store 710 MCC 172 constitute interrupt routines for the MCC or routines selected based upon information passed during handshaking.

MCC Operation

The exact details of the operation of MCC 172 will vary with the particular implementation of the MCC and the microcode program. However, a general description of execution in MCC 172 follows which will demonstrate the basic principles of operation within the present architecture in the context of embodiments illustrated in FIGS. 23A-C.

In operation, sequence controller 700 outputs a microcode address 702 to microcode store 710 in order to output a microcode control word which contains control words that direct the operation of other individual functional units within MCC 172.

The microcode control word 712 includes control word 712A which combines with output control signal 752A from execution address generator 750, to drive opcode generator 720 to generate opcodes which are output on opcode bus 178 to execution unit 130. The opcodes drive execution unit 130 to process a block of execution data originating in memory 800. In the event that multiple execution units are utilized, control word 712A and opcode generator 750 can be reconfigured to support opcode generation for the additional execution units.

In addition to controlling the algorithmic functions (adds, multiplies, etc.) of execution unit 130, the opcodes output on opcode bus 178 have the ability to reconfigure the data path of execution unit 130 under microcode control. For instance, execution unit 130 may be reconfigured from a 32 bit engine to a 16 bit engine for processing smaller operands or the execution unit could be configured to perform a four-cycle butterfly FFT for one DSP algorithm and reconfigured to perform a three-cycle butterfly FFT for a different DSP algorithm. Also, the pipeline of execution unit 130 can be reconfigured from, for example, a four deep pipeline for 32 bit operation to a two deep pipeline for 16 bit operation. Furthermore, the opcodes output on opcode bus 178 control the use of register files in execution unit 130 in order to support local storage within the execution unit.

The combined ability to control register file storage and pipeline operation in execution unit 130 results in the flexibility to achieve a wide variety of execution configurations, such as those shown in FIGS. 25 and 26. In a first configuration of execution unit 130, shown in FIG. 25, operands D1 and D2 are input to 8X8 multiplier 132 which generates a product which is output to register file 134 and arithmetic logic unit (ALU) 136. A first output signal from ALU 136 is also input to register file 134 and a second output signal is input to output latch 138. The output of register file 134 is also input to output latch 138. However, execution unit 130 can be reconfigured by opcodes received from opcode generator 720 into other configurations, such as that shown in FIG. 26, wherein the contents of register file 134 feedback into ALU 136.

The flexibility in execution configuration of the present invention permits a wide variety of arithmetic functions to be performed efficiently. In addition, the execution unit can be reconfigured for special non-arithmetic data paths which are useful to performing other forms of coding, such as data coding and decoding or compression and decompression.

At various points in the performance of an algorithm by execution unit 130, immediate data in the form of constants or coefficients may be needed for further processing. Control word 712B combines with control output 752B from execution address generator 750 and control output 772A from DMA address generator 770 to drive coefficient generator 730 to generate the processing constants and coefficients 732 that are required by some of the operations performed by execution unit 130. Control output 752B is included in the determination of coefficient output 732 because some algorithms require different coefficients at various points during execution of the algorithm. Control output 772A is included because coefficient requirements can change in anticipation of a swap of execution memory and DMA-I/O memory. Coefficient generator 730 can be a coefficient table or can function algorithmically to produce immediate data for execution unit 130.

Control words 712C and 712E control execution address generator 750 and

DMA address generator 770 respectively. Execution address generator 750 is similar to conventional address generators that are available to support address computation for a variety of functions. Execution address generator 750 can be an address generator which produces a single address signal or can be a device with the capability to output multiple addresses, i.e. multiple addresses for access to different rows in separate memory blocks. For instance, programmable address generators are presently available which are capable of supporting the address generation patterns for FIR, FFT and convolution algorithms for digital signal processing. Other programmable address generators are available which are suitable for performing the address sequencing associated with compression and decompression algorithms. Still other programmable address generators can be used which are suitable for the specialized address generation required for other types of algorithms, such as JPEG, MPEG, DVD or speech recognition algorithms.

In response to control word 712C, execution address generator 750 generates execution addresses which are output on execution address bus 752 and which are latched into address register 760 which is connected to memory 800 by address bus 162. The execution addresses output to memory 800 on address bus 162 access a block of execution data which are output on data bus 184 as operands to execution unit 130. Execution address generator 750 also generates the execution addresses which are used to write the result data from execution unit 130 back into memory 800. Portions of execution address bus 752 also function as control signals to other functional units within MCC 172. Portion 752A provides input to opcode generator 720, portion 752B provides input to coefficient generator 730, portion 752C is input to latch timing and control circuit 740, portion 752D is fed back to sequence controller 700 to determine the next microcode execution address, and portion 752E is output to configuration controller 790 in order to provide input to the configuration of memory 800.

DMA address generator 770 is also similar to conventional programmable address generators and may include the capability to reorder data to allow the data

to be processed more efficiently. For instance, if memory 800 is a DRAM device, then DMA address generator 770 can be instructed via control word 712E to reorder incoming data in memory 800 so that operands for execution unit 130 are located on the same data row line so that multiple accesses by execution address generator 750 to different rows of memory 800 are not necessary.

DRAM devices are typically constructed such that a single row address is common to several column addresses. When consecutive memory accesses involve data words having different row addresses then multiple address decode cycles must be performed to obtain the data words. If, however, the memory access pattern for an algorithm can be predicted, then the data can often be reordered in memory so that data words that will be accessed simultaneously or consecutively share the same memory row address. DMA address generator 770 can be instructed to perform this reordering as data is streamed into memory 800 in order to optimize later accesses to the data.

Similarly, DMA address generator 770 can be programmed to perform addressing algorithms, such as modulo, indexed or offset addressing, to reorder data in memory 800 prior to execution or after execution. Alternatively, DMA address generator 770 can be configured to operate under the direct control of a core processor, through an external bus interface mechanism such as bus interface 142 in FIG. 23B, which then provides the information necessary (i.e. source and destination addresses, block size, etc.) to direct the address generation for DMA or I/O transfers involving memory 800.

In response to control word 712E, DMA address generator 770 generates DMA addresses which are output on DMA address bus 772 and which are latched into address register 780 which is connected to memory 800 by address bus 782. The DMA address generation is performed in order to stream blocks of input data, i.e. data to be processed, and output data, i.e. result data, in and out of memory 800 via data bus 180. Portions of DMA address bus 772 also function as control signals to other functional units within MCC 172. Portion 772A influences coefficient generator 730, portion 772B is fed back to sequence controller 700,

along with other feedback controls, to determine the next microcode execution address 702, and portion 752C is output to configuration controller 790 in order to affect the configuration of memory 800.

Multiple address generators can be utilized to support the particular requirements of a specific application. For example, in the embodiment illustrated in FIG. 23C, memory 800 is reconfigured by the configuration control circuit 790, in response to control word 712F, to include an additional memory block C 190 in order to support the use of a second DMA port 2. An additional DMA address generator 775 driven by control word 712G is added to the architecture of MCC 172 to generate addresses for memory block C 190. A portion of DMA address bus 787 from DMA address generator 775 is connected to sequence controller 700 to influence control sequencing in microcode store 710. Similarly, multiple execution address generators can be added to support multiple execution units.

Latch timing and control circuit 740 receives an opcode control signal 178A, which is part of the opcode bus 178 output from opcode generator 720, and output control signal 752C, which is part of the execution address bus 752 output from execution address generator 750, in order to generate latch timing and control signal 174. Latch timing and control signal 174 synchronizes the operation of execution unit 130 with the transfer of data with memory 800 on data bus 184 by generating the timing and control for the circuitry at the interface between execution unit 130 and memory 800 which is discussed in greater detail below. In effect, latch timing and control signal 174 coordinates the transfer of operands from memory 800 into execution unit 130 with the output of address signal 162 to memory 800. Similarly, transfer of result data from execution unit 130 is coordinated with output of execution addresses on execution address bus 162 to memory 800, in order to store the results of the processing performed by execution unit 130, by latch timing and control signal 174.

Configuration control circuit 790 receives control word 712F, along with output signal 752E from execution address generator 750 and output signal 772C from DMA address generator 770, and generates memory configuration control

signal 152 , which controls the sizing, partitioning and swapping of memory blocks within memory 800, such as that described earlier with regard to FIGS 5A-C. For example, configuration control 790 can swap blocks in memory 800 in response to execution address generator 750 or DMA address generator 770 outputting an address predetermined by the microcoded program. In addition, reconfiguration signal 152 determines the position of boundary 136 in FIG. 23B and boundaries 136A and 136B, which may be fixed boundaries or reconfigurable (virtual) boundaries, to create memory block C 190 in the embodiment illustrated in FIG. 23C. In addition, there may exist situations wherein the particular algorithm being executed by MCC 172 does not require all of the memory space of memory 800, in which case unallocated memory space will exist which is not part of either memory block A 160 or memory block B 150 and which may be utilized to form memory block C 190 (which can also be used as general address space by the processor).

Once execution of a microcode program in microcode store 710 has commenced, sequence controller 700 receives control word 712D from the microcode store 710, along with output control signal 752D from execution address generator 750, output control signal 772B from DMA address generator 770, and execution status signal 704 from execution unit 130, which represent state information, and determines the next microcode execution address to output on microcode address bus 702 in the sequence of the microcode program in microcode store 710.

In addition to multiple DMA and execution address generators and multiple execution units, there can be multiple opcode generator (720) and configuration control (790) blocks and multiple pipe-line control and latch timing and control (740) blocks. Furthermore, other known techniques for optimizing the performance of MCC 172 can be applied, such as two level microcode structuring.

Also, the execution units utilized within the architecture can be specialized units for particular functions, such as MPEG code/decode, graphics, data

compression/decompression or other applications for which specialized execution units have been developed. Other execution units which are not specifically directed at data computation but which can directly implement compression/decompression or code/decode algorithms can be employed. One specific example is an execution unit directed at switching applications which controls decoding of the header information in data packets, stores the packets for retransmission toward their appropriate destination, and controls routing of the packets.

DRAM BANDWIDTH MATCHING AT MEMORY/EXECUTION UNIT INTERFACE

An example of a performance enhancement achievable with the highly flexible architecture of MCC 172 of the present invention is a high speed memory interface between memory block 800 and execution unit 130. Memory access times for DRAM technology are typically on the order of 20 to 80 ns., but can vary widely depending upon such factors as the fabrication process used to produce the DRAM or the column size of the DRAM array (e.g. the number of transistors coupled to the column lines). Execution unit execution stages, on the contrary, typically cycle on the order of 2 to 6.66 ns.. Memory access, therefore, can represent a severe performance bottleneck.

The latch timing and control along with the memory configuration control of the present invention, however, are capable of providing a high speed interface between memory 800 and execution unit 130 wherein multiple words are read from or written to memory in order to match the relatively slow access time of the memory 800 to the fast execution speed of execution unit 130. The high speed interface permits several words of memory 800 to be accessed in a single read or write operation so that the performance of execution unit 130 is not limited by the memory access time for memory 800.

FIG. 27 illustrates a variation on the architecture shown in FIG. 23B wherein memory 800 is constructed using DRAM technology and is divided into

N memory blocks MM1, MM2 through MMN which contain operands for execution unit 130. For purposes of demonstration only, memory 800 is assumed to have a memory access time of 40 ns. and execution unit 130 is assumed to have a per stage cycle time of 6.66 ns.. The access and cycle times herein recited are intended only as an example and not as a limitation upon the present invention. Other access and cycle time requirements will arise for devices fabricated with different technologies. In addition, it is assumed in the example of FIG. 27 that multiple memory blocks MM1-N take the place of memory block A 160 in FIG. 23B and are required to accommodate data which has not been reordered in memory 800 when the data was streamed into a DMA memory block in memory 800, such as memory block B 150, by DMA address generator 770. As a result, the memory words to be read out are not positioned in the same row of memory in a single memory block, and, similarly, the result data generated by execution unit 130, as determined by the algorithm under execution, cannot be written back to a single row of memory. This example is more complex than the case in which the data in memory 800 has been reordered to optimize access for execution unit 130.

Execution unit 130 is a pipelined execution unit operating with a cycle time of 6.66 ns., for this example, and which includes multiport local register files for feedback within the execution unit. A non-pipelined execution unit could also be utilized.

In order to interface the relatively slow memory access time to the fast execution time of the execution unit, a multi-word latch 820 is included which is sized under control of latch timing and control signal 174B from latch timing and control block 740 to compensate for the difference in memory access time and execution cycle time. For this example, multi-word latch 820 is configured for six words IW1-IW6 which are read into latch 820 in parallel, as shown in FIG. 28. When configured for a three stage execution pipeline, execution unit 130 generates a result in 20 ns. (3 stages X 6.66 ns/stage). The three stage pipeline structure of execution unit 130 is shown in FIG. 29, which also illustrates the content input words IW1-IW6 of multi-word input latch 820 and output words

OW1-OW5 of multi-word output latch 860.

Because the data in memory block A 160 is not all in the same row, under the assumption for this example that the data was not reordered when streamed into memory block A, selection logic 810 is needed to route the words read from memory into the appropriate position in multi-word latch 820. The selection logic 810 is controlled by latch timing and control signal 174A which is generated by latch timing and control circuit 740. The input words IW1-IW6 are obtained from memory blocks MM1-N and integrated by selection logic 810 for input to multi-word input latch 820. Thus, for example, data words IW1 and IW2 may be obtained from memory block MM1, words IW3-5 from memory block MM2 and word IW6 from memory block MMN. Address signals 162A, 162B and 162C, shown in FIG. 28, provide the addressing into memory blocks MM1-MMN along row lines of memory 800. Address signals 162A-C can be addresses generated by multiple execution address generators such as execution address 750 or execution address generator 750 can, itself, be a device which has the capability to generate multiple addresses.

Selection logic 810, under control of latch timing and control signal 174A, determines which sense amps in memory 800 to activate in order to route the words of data in memory 800 to the appropriate position in multi-word latch 820. As illustrated in FIG. 28, the latch timing and control signal 174A input to selection logic 810 results in the selective enablement of sense amplifiers within amplifier blocks 812, 814 and 816 to route a word from any of memory blocks MM1-N to an input word position IW1-6 in multi-word latch 820. (Note here that the amplifiers within amplifier blocks 812, 814 and 816 are intended to each represent multiple amplifiers which, in turn, amplify each bit line for a word of memory data.) Latch timing and control signal 174A controls the selection of operands from memory blocks MM1-N by selection logic 810 in synchronization with the operation of execution address generator 750 and opcode generator 720 and, by extension, the operation of multi-word input latch 820.

Multiplexor logic 830, under control of latch timing and control signal

174C, then selects and routes the data words in each of the word positions IW1-IW6 in multi-word latch 820 as operands into execution unit 130. Note that latch timing and control input 174C can reconfigure the size of the words in multi-word latch 820 to be 8, 16, 32 or more bits per word in response to opcode control signal 178A. Corresponding reconfiguration is simultaneously implemented in the memory blocks MM1-N, selection logic 810, multiplexor logic 830, execution unit 130, high speed latch 840, DMUX logic 850, multi-word output latch 860 and selection logic 870. In addition, MUX 830 can access the words IW1-6 in any order. Therefore, two levels of switching are achieved. First, selection logic 810 performs space switching by obtaining data from anywhere within memory blocks MM1-N. Secondly, MUX 830 performs time division switching by inputting input words IW1-6 in any order desired.

Execution unit 130 performs its computations on the operands provided by multiplexor logic 830 and outputs a result which is latched into high speed latch 840. The result is then routed by DMUX logic 850 into the appropriate position of multi-word output latch 860 (OW1-OW5) for write back to appropriate address in memory block MM1-MMN, wherein output selection logic 870 selects the memory destination of each output word under control of latch control signal 174H, or back into selection logic 810 in order to provide feedback.

Each stage 130A-C of execution unit 130 is running at a clock of 6.66 ns. (20 ns. divided by 3). IW1 and IW2, for example, are multiplexed and clocked into 130A at the beginning of a first 6.66 ns. period. Then, the result from execution in 130A during the first period is clocked into 130B in the following 6.66 ns. period and the result from 130B during the second period is clocked into 130C at the beginning of a third period such that the result of the operation on IW1 and IW2 (IW1 X IW2) is stored in position OW1 of output latch 860 after a three pipe stage delay of 20 ns.. Subsequently, the product of IW1 X IW2 is fed back and multiplied by IW3 in 130A, with the next result being stored in position OW2 of output latch 860 after 34 ns.. The operand operations can be summarized as follows:

(1) IW1 x IW2 = A -> OW1; 20 ns.

(2) A x IW3 = B -> OW2; 20 ns.

(3) B x IW4 = C -> OW3; 20 ns.

(4) C x IW5 = D -> OW4; 20 ns.

(5) D x IW6 = E -> OW1; 20 ns.

where A-E represent the feedback values which loop back into 130A at the top of the pipeline.

After 80 ns.,the input latch 820 is free to be reloaded with another set of operands from memory blocks MM1-N because memory word IW6 has entered the pipeline. Similarly, after an initial delay due to results percolating through the execution pipeline, output latch 860 will fill every 80 ns. and be ready for a write back to memory 800 to occur. Once the pipeline is full, during each 80 ns. cycle, six words of data are read from memory 800 and input into input latch 820, 40 ns. later, for example, five words of result data in output latch 860 are written back to memory 800 and, 40 ns. after the write back, another six words are read from memory 800 and input into input latch 820. Meanwhile, operands are being continuously processed by execution unit 130 for output to output latch 860, which may require an additional temporary store positioned between DMUX 850 and output latch 860 to accommodate the results output from execution unit 130 during the 40 ns. write back cycle.

FIG. 30A illustrates a sample timing diagram for the clock signals within latch timing and control block 740, for the present invention. An example of an embodiment of latch timing and control block 740 is shown in FIG. 30B to demonstrate the generation of the latch timing and control signals 174A-H in the embodiment illustrated in FIGS. 27 and 28.

An oscillator 742 generates a 6.66 ns. period clock CLK3 which enters counter 744A and is divided into a 20 ns. clock CLK2. The 20 ns. clock signal CLK2 is then divided by counter 744B to generate an 80 ns. clock signal, CLK1. Write back of result data from multi-word output latch 860 to memory blocks MM1-N occurs at 80 ns. intervals that are staggered by 40 ns. from the read cycle

and are illustrated as the dotted pulses in CLK1. The resulting clock waveforms are shown in FIG. 30A. As noted above, these clock cycle times cited are exemplary and other operating cycles may be utilized.

Latch timing and control signals 174A-H are controlled by and must be synchronized with opcode output signal 178A and execution address output signal 752C. Signals 174A-H provide both timing information and control. Signal 174A determines which data words within memory blocks MM1-N are selected by selection logic 810 and also to select them in synchronization with CLK1, in this example. Signal 174B controls the width of multi-word input latch 820 and also the latching of the data selected by selection logic 810 synchronously with CLK1, in this example. Signal 174C controls the order with which MUX 830 inputs data words IW1-IW6 into execution unit 130 in time with CLK2, in this example. Signal 174D drives the execution unit at CLK3. Signal 174E drives high speed latch 840 at CLK2 to latch results as they are output from execution unit 130. DMUX 850 is controlled by signal 174F to route the output word in high speed latch 840 to temporary storage or a word position in multi-word output latch 860 in time with CLK2. Signal 174G drives multi-word output latch 860 to receive a result word in time with CLK2 and write the result data back to memory 800 in time with CLK1. Signal 174H controls where in memory blocks MM1-N each result word OW1-6 in multi-word output latch 870 is written back in time with CLK1.

The reconfigurability of the various blocks in FIG. 27 also requires that latch timing and control signals 174A-H be flexibly reconfigurable. Signal control blocks 746A-H, in FIG. 30B, generate signals 174A-H respectively under the control of opcode signal 178A and execution address signal 752C. A variety of timing and control sequences are obtainable by altering the codes input by opcode signal 178A to signal control blocks 746A-H in order to match the flexibility and reconfigurability, under the control of MCC 172, of memory 800 and execution unit 130. The circuit of FIG. 30B can be extended to include further signal generation blocks to accommodate, for instance, timing and control signals to

individual stage elements internal to execution unit 130.

FIG. 29 illustrates one configuration of the internal stage elements 130A, 130B and 130C of execution unit 130 in relation to the memory and execution unit interface circuitry discussed above. As discussed above, latch timing and control signal 174B drives input latch 820 and signal 174F drives output latch 860 in reading data from and writing data back to memory 800, respectively, at the 80 ns. intervals of CLK 1. Signal 174C drives MUX 830 to route an operand from input latch 820 to stage element 130A of execution unit 130 every 20 ns. in time with CLK2. Signal 174E drives high speed latch 840 to also capture the result output from stage element 130C every 20 ns. in time with CLK2. Signal 174D drives the internal pipeline stage elements 130A, 130B and 130C of execution unit 130 in time with CLK3 to process the operands supplied by MUX 830.

One advantage to the present architecture is that most of the MCC 172 circuitry runs at the relatively slow 80 ns. clock speed of CLK1. Memory blocks MM1-N are clocked every 40 ns. by the alternating read and write cycles described above. Selection logic blocks 810 and 870 operate at the CLK1 rate. MUX 830, high speed latch 840 and DMUX logic 850 run at the 20 ns. clock speed of CLK2 (in this example, but are capable of operating at higher clock speeds). And only the pipeline stages 130A, 130B and 130C of execution unit 130 run at the highest clock speed of CLK3. Because most of the MCC 172 circuit runs at the slower clock speed, the power dissipation of the circuit is relatively low.

As discussed above, the reconfiguration of the datapath in execution unit 130 is accomplished by the opcodes 178 generated by opcode generator 720. The control signals 174A-H generated by latch timing and control block 740 are determined in part by opcode signal 178A. Therefore, the datapath can be dynamically reconfigured along with the control and timing by the output of opcode generator 720.

For instance, stages 130A and 130B can be reconfigured such that they require a slower clock cycle, for example 2 X 6.66 ns., in order to operate. The

opcode signal 178A then combines with execution address portion 752C, as shown in FIG. 23B, to modify the output 174 from latch timing and control block 740. As shown in FIG. 30B, programmable counter 744C receives opcode signal 178A and execution address signal 752C and produces a clock signal with a period of M X CLK3, which is 2X 6.66 ns. in the present example, which is output to stages 130B and 130C of execution unit 130. As a result, the operational feedback for results feeding back from 130C to 130A becomes 5 X 6.66 ns. (1 X 6.66ns. for 130A, 2 X 6.66 ns. for each of 130B and 130C). Therefore, a second programmable counter 744D is included which divides the basic 6.66 ns. clock signal by a value N, which is 5 for the example shown, determined by the control inputs 178 and 752C.

The example of FIG. 29 illustrates how the pipeline and local storage timing can be reconfigured under the control of the opcode generator 720 and latch and timing control 740. The local storage and feedback internal to execution unit 130 can take a variety of forms in addition to the closed loop feedback shown in FIG. 29, such as a circuit which includes a multi-ported register file for active storage in the feedback path, as shown in FIG. 31. Note that the multi-ported register file in FIG. 31 can also be positioned at the top of the pipeline between MUX 830 and stage element 130A.

To demonstrate the flexibility available through reconfiguration of the datapath of execution unit 130, a reconfigured execution unit is shown in FIG. 32 which includes a multiplier block 130A, an adder 130B and a multiported register file 130C. Multiplier 130A can, for example, be configured as two single stage 16 bit multipliers including latches or as a single four layer 32 bit multiplier composed of two 16 bit multipliers which are cycled twice. Adder 130B can be configured as two 16 bit adders or as a single 32 bit adder.

As noted above, in the configuration shown in FIG. 27, the selection logic blocks 810 and 870 are necessary to accommodate the data in memory blocks MM1-N which, in the example illustrated, was not aligned by DMA address generator 770 as the data was streamed into memory 800. The embodiment of

FIG. 27 is complex and demonstrates the highest degree of flexibility because it can control the reordering of data to handle algorithms wherein the data cannot be reordered by DMA generator 770 for direct access to a group of data words in a single row of memory 800 during read or write back. It is important, in the embodiment of FIG. 27, that high speed latch 840 provide sufficient multiported temporary storage to support the highly pipelined data path of execution unit 130 such that the memory to execution unit interface is compatible with DRAM memory operating at cycle times of 20 to 80 ns.. High speed latch 840, in the embodiment of FIG. 29, is the same register used to provide feedback in the pipelined datapath of execution unit 130, but it can also be deployed as a separate unit. It should also be noted that selection logic block 810 and multi-word input latch 820 can, in some applications, be replaced by a multi-ported register file. Similarly, selection logic block 870 and multi-word output latch 860 can also be replaced by a multi-ported register file.

However, DMA address generator 770 can be programmed to provide sequencing and control for aligning data for optimum processing by the execution unit 130 of algorithms which do permit reordering of the data. For instance, in DRAM memory technology, placing in the same row data which is to be accessed simultaneously (i.e. all six words, in the example, to be read into multi-word latch 820), the selection logic 810 and 870 for reading and write back, respectively, can be eliminated and memory blocks MM1-N consolidated into a single memory block to simplify the circuit and improve the effective access time of the memory to execution unit interface.

In addition, DMA address generator 770 can be programmed to reorder data after processing by execution unit 130, i.e. reorder the data while streaming the data out of memory 800. As a simple example, bytes of result data in memory 800 can be "swapped", the higher order and lower order bytes interchanged to match another processor's data format, as the bytes of data are streamed out of memory 800 to the core processor or other destination. More sophisticated and complex reordering is also possible.

Data Reordering

An example of the reordering discussed above is the reordering required to perform an FFT algorithm. The following is an example of a 16-point radix-2 DIT FFT implemented in MCC 172 in the configuration illustrated in FIG. 23A. First, execution is initiated, by means of one of the mechanisms for execution initiation in MCC 172 described above, which causes sequencer 700 to output, on microcode sequence signal 702, the address in microcode store 710 of a 16-point radix-2 DIT FFT microcode program. When sequencer controller 700 issues this first micro-code address for the FFT routine in micro-code store 710, micro-code store outputs a full micro-code word 712.

Microcode control word 712D is received by sequence controller 700 to set up for execution of the next microcode command in the FFT sequence, while microcode control word 712F is received by configuration control block 790 in order to determine the memory configuration control signal 152 which is output to memory block 800 to configure the memory for the FFT. In the present example, the data to be processed by execution unit 130 will be initially be loaded into memory block B 160 and will be prescrambled prior to switching memory block B 160 with memory block A 150.

Micro-code control word 712E entering DMA address generator 770, provides the following information to DMA ADDR Generator 770:

- Starting address for location of data in DMA memory block is X.
- Sixteen data points starting at X are to be scrambled for FFT.

The DMA address generator 770 then reorders the data as follows:

BEFORE SCRAMBLE		AFTER SCRAMBLE	
Location	Data	Location	Data
X	(0)	X	(0)
X+1	(1)	X+1	(8)
X+2	(2)	X+2	(4)
X+3	(3)	X+3	(12)
X+4	(4)	X+4	(2)
X+5	(5)	X+5	(10)
X+6	(6)	X+6	(6)
X+7	(7)	X+7	(14)
X+8	(8)	X+8	(1)
X+9	(9)	X+9	(9)
X+10	(10)	X+10	(5)
X+11	(11)	X+11	(13)
X+12	(12)	X+12	(3)
X+13	(13)	X+13	(11)
X+14	(14)	X+14	(7)
X+15	(15)	X+15	(15)

The numbers in parentheses represent the offset location from X where the data was originally located.

These scrambling operations by the DMA address generator 770 involve reading X+8 (8) and storing in a temporary buffer or location and then writing X+1 (1) into X+8, this is followed by writing temporary location (TR) data (8) into location (X+1). The next step is storing X+4 (4) into the temporary buffer via read/write cycle followed by reading data located in X+2 (2) and writing into

(X+4) 2. This operation is followed by reading TR data (4) and writing it into location X+2. This process continues under control of the DMA address generator 770 until all 16 points of data have been reordered. Alternatively, the microcode program in microcode store 710 can be suitably structured to program the DMA address generator 770 to reorder the data in the manner shown above as the data is streamed in from an external source, such as through Data I/O port 180.

When MCC 172 includes the hardware shown in the embodiment of FIG. 27, however, reordering by DMA address generator 770 may not be necessary or advantageous since selection logic 810 and multi-word input latch 820 can perform a wide parallel read from the execution memory, memory blocks MM1-N in FIG. 27, and then MUX 830 can reorder the data within input latch 820 on the fly by taking operands from the latch in the order required by the FFT algorithm. This feature of automatic (on the fly) reordering eliminates the need for prescrambling by the DMA address generator prior to swapping memory block A 160 and memory block B 150 before processing by execution unit 130. On the fly reordering is useful in cases where the data is not amenable to the reordering required by the processing algorithm prior to execution. Reordering on the fly also has the advantage that it can typically be performed by MUX 830 at much higher speeds than reordering by DMA address generator 770 which is subject to the relatively slow memory access times of memory 800.

Built-In Self Test and the Memory Centric Controller 172

Another advantage to the present architecture is that an extensive BIST can be performed on the memory array 800, execution unit 130, and MCC 172 itself. The ability of MCC 172 to reconfigure the data path and timing, via opcode generator 720 and latch timing and control block 740, of execution unit 130 allows a worst case data path, or a variety of different data path configurations, to be selected for testing. In addition, latch timing and control block 740 can be configured to test execution unit 130 at an elevated clock rate. Instead of driving execution unit 130 at 6.66 ns., for example, the opcode signal 178A can be

configured to operate execution unit 130 at 5 ns. timing under control of a microcode BIST loaded into microcode store 710. The elevated test clock rate enables the BIST to compensate for the differences between room temperature wafer testing of devices and higher temperature packaged testing devices. The microcode BIST is able, due to reconfiguration to a higher clock speed, to detect defective devices that will fail a BIST executed at normal clock speeds after the devices are packaged.

There is a special BIST microcode downloaded which is performed in order to do the self-test. The downloaded BIST microcode supports the selection of a worst case path in execution unit 130 through the microcode control word 712A output to opcode generator 720. For instance, instead of configuring the execution unit elements to be three or four stages deep, opcode signal 178 configures execution unit 130 to have two data paths which are two stages deep which each operate at 5 ns. in order to obtain a stress performance of the execution unit elements. The worst case path is run it at a faster clock rate during wafer testing than is the clock rate that is normally utilized during the high temperature packaged testing.

Another advantage of BIST in the present architecture is that the BIST microcoded can be downloaded from test probes at a low speed but the BIST microcode executes at high speed. Similarly, the test results and can be subsequently transferred through the test probes at a low transfer rate. Testing of an MCC based device is therefore not restricted by the electrical limitations (i.e. inductance, capacitance, noise interference, etc.) of the test probe interface to the MCC based device.

Also, using the test microcode, the entire memory block 800 can be minutely checked because the test microcode controls configuration control block 790 in order to perform a variety of memory configuration set-ups to check the function of the sense amps within memory array 800. Memory array 800 can be configured to have multiple memory blocks, such as block 150 and 160, which can be checked by switching the blocks back and forth. Also, memory test

patterns, such as a checkerboard pattern, can be written into memory and read back out in order to check the function of individual memory cells in memory array 800. Defective memory elements can be located and information relating to the location of defective cells can be output so that a determination can be made of what circuitry to burn in through a laser fuse in order to replace the defective cells with redundant memory elements which are typically included when memory arrays are fabricated.

The defective memory element information can be output as a status word or other data which can be output through a dedicated test port or through a status word in memory array 800 which is accessible through the data interface ports, such as data I/O bus 180 or external bus interface mechanism 142, which are already provided by the present architecture. Compilation and output of status information is performed, again, under control of the microcode self test. The result is a high powered memory based device which can test itself and which can reconfigure itself to obtain worst case or special case testing.

Power Management by MCC 172

The MCC 172 also has the capability to manage the power dissipation of execution unit 130 and the components of MCC 172. As was described above with respect to latch timing and control block 740 and FIG. 30B, the clock speeds used to drive execution unit 130, as well as the other functional blocks shown in FIG. 27, can be adjusted under control of opcode signal 178A which, in turn, is determined under control of microcode control word 712A input to opcode generator 720. For instance, as described above, the clock speed for execution unit stage elements 130A-C can be modified to be 2 X CLK3 resulting in an effective clock cycle, in the example shown, of 13.33 ns. and cutting the operating frequency in half. The relation of power dissipation to operating frequency is $P=CV^2f$. Therefore, doubling the clock cycle cuts the power dissipation in half.

Thus, because the microcode program for the algorithm in microcode store 710 has embedded in it the control information to modify the speed of data

transfer through input latch 820, MUX 830, high speed latch 840, DMUX 850, execution unit 130 and output latch 860, the microcode also has the capability of controlling the power dissipation of these circuits by reconfiguring the clock cycle times of the circuits. For instance, as described in the example above, MUX 830, under the control of latch timing and control signal 174C, selects an operand from input latch 820 for input to the pipeline of execution unit 130 at the 20 ns. intervals of CLK2. In fact, operands can be selected at the 6.66 ns. clock rate of CLK3 constituting a threefold increase in the operating frequency, and therefore the power dissipation, of MUX 830. Thus, by moving data from latch 820 into execution unit 130 at 20 ns. intervals, a substantial improvement in power dissipation is achieved.

In addition, the microcode program can include multiple subroutines which support different levels of performance of an algorithm and, consequently, also support different levels of power dissipation. The microcode program can be structured to select a microcode subroutine which configures execution address generator 750, opcode generator 720 and latch timing and control circuit 740 for slower operation when fast performance is not required. Power dissipation is reduced in the execution address generator 750, the opcode generator 720 and latch timing and control 740 by the microcode selected for an algorithm. The opcode generator block 720 and latch timing and control block 740 have their power reduced by microcode that generates slower signal frequencies in latch timing and control signals 174A-H to the data path and reducing the requirements of the opcode output 178 (updates of the execution unit and latches occur less frequently which decreases the operating frequency). This also means less power is dissipated in the latch timing and control unit 740 itself. Thus, there is power reduction achieved within the three major block levels described above. Also note that the reconfiguration of the data path will also result in reduction of power dissipation. The important consideration here is that the microcode subroutine can effect the power dissipation of algorithmic calculations by determining the timing rates for latches and timing signals for pipe-lines in the execution unit, and

indeed, the nature of the reconfiguration of the execution units themselves.

The above power reduction technique, in addition to being enabled through a specific piece of micro-code, can also be enabled via branching into a segment of microcode that enables less power dissipation. For example, for a given algorithm there may be more than one micro-code subroutine which would support the algorithm of different levels of power dissipation and speed. Thus, through a branching option, this selection would be determined. For example, when more then one algorithm is being processed concurrently through the execution unit 130, it may be necessary to utilize the faster speeds in the latch timing and control signals 174A-H and faster issuance of opcodes in opcode signal 178 in order to meet the throughput requirements of the algorithm. However, when only one algorithm is being processed in execution unit 130, slower latch timing and control and issuance of opcodes can be accommodated. An example would be executing an FIR and FFT through parallel data paths in execution unit 130 with the goal of providing maximum throughput versus executing an FIR alone in which the highest throughput was not necessary.

High-speed Branching

The reduction of power dissipation by arranging branching, as discussed above, is not the sole reason for utilization of a branching mechanism in this architecture. There are two levels of branching in the present architecture. The first level of branching is at a slower speed and occurs through the operation of sequence controller 700 and sequences at a rate equivalent to approximately the read/write cycle time of memory array 800, which is typically a DRAM. Power dissipation in sequence controller 700 is reduced because the slower operational speed allows for minimal size device transistors, since the performance of sequence controller 700 does not have to perform at a high clock rate, and simpler levels of chips interconnect can be utilized.

As illustrated in FIG. 23C, the execution unit feedback signal 704 from the

execution unit 130, as well as the feedback signals 752D, 772B, and 787B, provide feedback for creating a branch in the microcode sequence under the control of sequence controller 700. Thus, branching in the sequence controller 700 can be achieved by standard methods and, as mentioned above, can be supported by more then one level of micro-coding standard methods. This high-level branching method can support the power reduction techniques described earlier as well as bringing various micro-code subroutines into operation.

However, to accomplish a high-speed branch, additions need to be made to the architecture which are illustrated in FIG. 33. The execution unit 130 provides feedback control information signals 704A to opcode generator 720 and 704B to latch timing and control circuit 740. Feedback signal 704 continues to be connected to sequence controller 700. The feedback of signals 704A and 704B is necessary to enable a high-speed branch to occur within the processing performed within the execution unit 130 as opposed to the lower speed branch associated with sequence controller 700. Specifically, the high-speed branch arranges modifications to the operations which take place within the circuitry illustrated in FIG. 27, excluding memory. The circuit blocks 810, 820, 830, 840, 850, and 860 of FIG. 27 indicate a requirement to relate to high-speed calculation of the algorithm coupled with optimizing or reordering the operands which feed into the pipeline of execution unit 130 to accommodate the read/write issues associated with memory 800 (i.e. DRAM) or algorithms requiring descrambling of data, e.g. reordering input data to do an FFT as described above.

An example of fast branching involves executing a first FIR algorithm FIR A followed by a branch to one of a second FIR algorithm FIR B and a convolution algorithm CONV B, which is illustrated in FIG. 34. The data required for implementing FIR A, which involves a small number of data points, is loaded into multi-word input latch 820 (or a multi-ported register file which takes the place of blocks 820 and 830). After FIR A completes, a fast branch occurs to either FIR B or CONV B depending on a branch condition resulting from FIR A which feeds back to opcode generator 720 via feedback signal 704A and latch timing and

control block 740 via feedback signal 704B, as shown in FIG. 33.

Note that the high speed branch to either FIR B or CONV B is achieved by opcode generator 720 and latch timing and control block 740 in response to the feedback signals 704A and 704B. Opcode generator 720 issues the next opcode on opcode output signal 178 and outputs opcode control signal 178A to latch timing and control block 740 to drive the next latch timing and control signals 174A-H in order to execute either FIR B or CONV B, depending upon the branch feedback condition reflected in feedback signals 704 A and 704B.

Speculative Execution in MCC 172

The present architecture also has the capability to perform speculative execution. Expanding upon the example discussed above in regard to FIG. 34, assume that FIR A, FIR B and CONV B are long algorithms wherein each of the FIR operations require a large number of operands, thus resulting in the need to perform multiple read and write cycles to memory 800 (whereas the previous example was limited so that the algorithms could be executed within execution unit 130 without additional memory accesses). In the present example, speculative execution can be accomplished by doing FIR A, FIR B and CONV B in parallel.

For instance, execution unit 130 can be configured, as described above, to have two parallel data paths wherein FIR A and FIR B are calculated in the first data path and CONV B is calculated in the second data path. Note that since these operations require multiple accesses to memory 800, execution address generator 750 (or multiple address generators) provides address generation for the processes of all three algorithms running in parallel. This can be accomplished by a high level address generator capable of generating addresses for multiple processes (algorithmic memory addressing) being addressed concurrently or by the use of smaller address generators where each smaller address generator handles one process. The result of address generation is that operands are obtained from memory 800 and input to execution unit 130 for FIR A, FIR B and CONV B in

parallel. The results from FIR B and CONV B are held in temporary memory locations until FIR A completes thereby setting the branch condition. Based upon the branch condition from FIR A, the temporary memory location for either FIR B or CONV B is switched into the final result segment of storage in memory 800, e.g., switched into working memory through reconfiguration of the memory. Thus, reconfiguration of memory 800 also comes into play in obtaining speculative execution in the present architecture. The process flow of the present example is illustrated in FIG. 35.

The configuration of memory 800 into working memory and temporary memory locations 1-N, as shown in FIG. 36, is accomplished by configuration control block 790 under the control of the microcode program in microcode store 710. Reconfiguration of memory 800 will occur after the branch from FIR A to either FIR B or CONV B in the present and prior examples.

Note that it is possible, in the event that FIR A completes and the algorithm FIR B or CONV B that is selected based upon the results of FIR A has not completed execution, to reconfigure execution unit 130 to devote more processing resources to completion of the selected algorithm. At the same time, memory 800 can be reconfigured to couple the temporary memory location for the selected results into the working memory containing the results of FIR A.

Further note from the present example that reconfiguration of memory 800 together with speculative execution combine to accomplish higher execution speed for computational algorithms.

Fig. 37 is a simplified block diagram showing the interconnection of a processor, such as a known microprocessor 10, coupled to a conventional memory 12 via a bus 14 ("Z"). A memory centric computing engine 16 of the type described in the two prior applications identified above also is coupled to bus 14 for transfer of address, data and control information among these three system components. A system of this type can be implemented in one or more integrated circuit, although the present state of the art provides only limited memory space on board a typical CISC microprocessor. The memory centric computing engine

16 is intended to include a substantial amount of memory, for example DRAM, as explained in the prior application. The particular arrangement of address, data and control signals, and interfacing or "handshake" protocols varies with the various different microprocessors and bus architectures. As described below, the interface apparatus and methodologies of the present invention are highly flexible so as to provide compatibility with various interfaces without hardware modification.

Fig. 38 shows the memory centric engine 16 in somewhat greater detail. In this illustration, a memory space 40 is shown as being partitioned into four blocks of memory identified as a microcode memory 20, execution memory 22, I/O memory 24 and available processor memory 26. As described in the parent application, the memory is configurable so as to form one or more blocks of memory, each sized as may be required for a particular operation. One or more execution units, for example execution unit 28, is coupled to the execution block of memory 22 for transfer of data therebetween. The memory centric computing engine is controlled by the memory centric controller (MCC) 30, as described in detail in the related application entitled MEMORY CENTRIC CONTROLLER. For example, it controls configuration of the memory 40, and provides address generation for each block of memory. Buses 32, 34 and 36 in this drawing generically represent address, data and control lines. Finally, both the I/O memory block 24 and processor memory block 26 have data ports coupled to internal bus 38 which in turn is connected to the external bus 14. Operation of this architecture is described in detail in the memory centric controller application, except for the interface to the external bus 14.

Fig. 39 is a simplified block diagram illustrating connection of the engine 16, more specifically the MCC controller 30, to a flash, ROM, or other non-volatile memory 50, via path 52. The flash or ROM storage 50 is used to store microcode executable in the MCC. One mechanism for loading microcode is to arrange the MCC so that, upon initialization, it downloads microcode directly from the external device 50. Further description of operation of the system is provide below following an overview of the hardware arrangement.

Fig. 40 is a simplified block diagram illustrating one implementation of the memory centric computing engine that includes a bus interface controller ("BIC") 60. Fig. 40 shows the memory 40 coupled to the MCC controller 30 as described previously. Here, the BIC 60 provides for interfacing the memory centric engine to a standard or customized bus interface. The external bus 38 can include various control and address signal lines, but at a minimum includes at least one data line (serial interface) though it will often include a plurality of data lines for bite-wide or word-wide data transfer. The MCC 30 provides control signals 62 to the BIC 60 to synchronize communication from the BIC to the I/O memory block 24. Data is transferred over bidirectional, internal data bus 66 between the BIC and the I/O memory block 24. The bus interface unit (BIC) provides control signals 64 to the MCC so as to provide status information. For example, such information can include the status conditions of a buffer memory in the BIC. In addition, fast address decoding (further explained later) can be implemented in the BIC and resulting control signals provided to the MCC. The bus interface controller, or BIC, further includes a buffer memory 68 which is used for buffering data in connection with data transfers between the external bus 38 and internal data bus 66. The status conditions provided in connection with loading and unloading the buffer memory 68 enable the MCC to evaluate when a communication from the I/O memory block to the bus interface should be initiated or, conversely, a communication from the bus interface to the I/O memory block. (These status signals are distinguished from the status word or block used for communication between the MCC and the host processor as explained later.) Preferably, the buffer memory 68 comprises SRAM or other high speed memory in order to accommodate high speed data transfer to and from the external bus 38 as further described later.

Fig. 41 is a block diagram of the computing engine of Fig. 40 showing greater detail of the bus interface controller 60. In this implementation, the BIC 60 includes a buffer memory, comprising SRAM block A 70 and SRAM block B 72 together forming the buffer memory. SRAM memory is preferred for its high

speed as mentioned above. A single block of memory could be used. However, it will be advantageous in many applications to use two or more blocks or memory which can be "swapped" as described in detail in the parent application entitled SHARED, RECONFIGURABLE MEMORY ARCHITECTURES FOR DIGITAL SIGNAL PROCESSING. Specifically, one block of SRAM, for example 70, can be coupled to the external bus 38 to transfer data (bidirectional). Concurrently, the second memory block B can be coupled to the internal data bus 66 for transferring data into the memory 40 for execution or to fetch results. After a given block of data has been transferred, the buffered memory SRAM blocks A and B are swapped, i.e. block A is coupled to the internal data bus 66, and block B is coupled to the external bus 38 to begin transferring a subsequent block of data.

Data transfer via the internal data bus 66 requires matching bandwidths of the fast SRAM 72 with relatively slow DRAM 40. One technique for doing so is to configure the I/O block of DRAM 40 so that each row is an integer multiple, for example 8 or 16 times, the word size, i.e. the number of bits of data words in the buffer SRAM 70, 72. Referring to Fig. 44, demultiplexer circuit 84 directs successive words from bus 66 into respective locations in a wide high speed latch 86. Accordingly, it takes multiple reads of the SRAM buffer, say 16 cycles, in order to fill the wide latch 86. When the latch 86 is filled, the wide word is transferred in a single write cycle into the wide I/O memory block in DRAM 40. These operations are controlled by the MCC 30 in a manner analogous to the execution unit bandwidth matching described in the related case identified above. In fact, multiplexer, demux and latch hardware for reading and writing the DRAM 40 can in some configurations be shared as between the internal data bus 66 and the execution unit 28. In this manner, the transfer rate over internal bus 66 can essentially be matched to the bandwidth of the external bus 38. Similarly, in order to read data from the DRAM, a wide word of data can be written into the wide latch in a single read cycle, and then individual bytes or smaller words sequentially routed through a multiplexer to the data bus 66 for output at a correspondingly higher transfer rate to the host.

However, a read operation still suffers from the relatively long latency of a read cycle before the first word of data becomes available. This problem can be addressed through the selective use of SRAM cells in DRAM array, or at least in the I/O memory block. For example, the first word or two of each row of the memory could be implemented in SRAM rather than DRAM. See Fig. 42. In this way, when a given row of the memory is accessed, the first word or two becomes available for transfer to the internal data bus at SRAM access speed. While that initial data is transferred, the rest of the row is latched in the wide latch described above and then sequentially transferred onto the bus in smaller units.

The use of SRAM cells in part of the DRAM array can be especially useful in telecom applications such as a router, where the format of the data, e.g. packet size, is predetermined and fixed. In that case, the engine can be configured so that every memory access in on an even row boundary. Accordingly, the data first accessed, i.e. the first word of the row, is always an SRAM word, ergo latency is minimized.

The size of each DRAM row is not critical, and depends upon the particular application. Preferably, a relatively wide word size is implemented to allow greater data transfer bandwidth as explained below.

DRAM Memory with SRAM Interface

The methods an apparatus described above can be used to provide for DRAM storage having SRAM interface speed. Here we use the terms "DRAM" and "SRAM" merely as illustrations or shorthand to indicate on the one hand a relatively dense but slow memory (DRAM) and, on the other hand, a less dense but faster random access memory (SRAM). The bandwidth matching and buffering techniques described herein can be applied to other memory technologies. Referring to Fig. 43, bus 99 represents a standard SRAM interface, comprising data signal paths coupled to a data port of an SRAM array 100 and control signals 112 which, in this case, are directed to a bus interface controller

106. The interface 99 can conform to standard memory interface, for example, comprising data lines, address lines, address strobes, read/write enable signals, and the like.

A data port of the SRAM 100 is coupled to a data port of a relatively large DRAM array 102 via an internal data bus 104. The bus interface controller 106 provides addressing and control signals 108 to the SRAM 100 and provides synchronization handshaking with the MCC. The MCC, in turn, provides address generation and control signals 110 to the DRAM array 102. Thus the SRAM can interface with the bus 99 at relatively high speed, while the DRAM array 102 provides relatively large and inexpensive storage. Of course, this system will not provide true random access to any location in the DRAM array at SRAM access times. Nonetheless, for many applications, it will be highly advantageous. A ROM or other non-volatile storage or writable control store 116 can be provided to store a microcode for execution in the BIC 106. Alternatively, the BIC can be hard wired to implement a particular application. As another alternative, microcode can be downloaded into the BIC or into the DRAM using the techniques described herein in the context of the memory centric computing engine. The architecture of Fig. 43 thus provides a pseudo-static memory device useful in many applications.

Fig. 44 shows portions of the apparatus of Fig. 43 in greater detail. Here, the bandwidth matching hardware mentioned above is shown explicitly, During a read operation, a wide row of the DRAM 40 is written into a wide, high-speed latch 86. Individual portions or words of data from latch 86 are directed through demultiplexer 84 onto the internal data bus 66 and transferred into the SRAM 70, 72 one at a time. Conversely, for a write cycle, the input data is directed through MUX multiplexer 90 into a demultiplexer 89 which in turn directs individual words of data into successive locations in a wide latch 88 until the latch is filled. Each time the latch is filled, the wide word is written broadside into the DRAM 40. Where this circuit is implemented in a memory alone, the multiplexer 90 is unnecessary. In the context of a computation engine, multiplexer 90 can be used

for switching data into the DRAM from an execution unit.

In an alternative embodiment, the execution unit can be coupled directly to the internal data bus 66 for streaming data into execution from the buffer memory 68 for high speed operation, essentially bypassing the DRAM. In this implementation, the MCC configures the execution path accordingly, and provides synchronization and control signals to the BIC which, in turn, provides address generation and control to the SRAM buffer. Alternatively, the op code generator and latch timing and control block of the MCC, described in the related MCC application, can be extended to provided address generation for accessing the SRAM buffer. This mode of operation can be enabled through microcode just as other operations described herein.

OPERATION

1. Supply microcode.

First we describe how microcode is supplied to the MCC using the architecture described above. It can be done in several ways, depending on the particular application. In one example, the processor 10 executes program in external memory 12, and under that program control loads microcode into the engine via bus 14 (Z) — using the well-known DRAM interface mentioned previously.

Alternatively, the available processor memory 26 located in the engine could also be source of program running on the processor 10 to supply the microcode to the engine via bus 14 and memory block 26.

Another example is a combination of running on memory 12 and available processor memory 26 so as to support the generation of microcode. In other words, the processor program execution generates the microcode.

A further method of supplying microcode to the MCC is to load it into a microcode block of memory 20. In other words, predetermined microcode is passed via the Z-bus into the micro-code space 20 under processor control while

the MCC is passive. In this case, the addressing is handled by the processor as indicated in Fig. 45. As illustrated in the Fig. 45, a first address latch can be used for processor addressing of the microcode memory space to download microcode, and a second address latch is enabled during execution of the microcode. The outputs of the address latches can be wired-OR together. See Fig. 45. This process is initiated by the MCC is response to decode of a predetermined (phantom) address that implies or triggers this operation, i.e. a memory mapped operation. A hard-wired decode (or a dedicated pin) can be provided for power-up reset. Another mechanism for loading the microcode into the MCC is on initialization — it automatically (as part of initialization) reads in code directly from a flash or ROM type of device 50, external to the engine, as illustrated in Fig. 39. By "engine" we mean MCC controller together with a memory subsystem. See Fig. 39. The application shown in Fig. 39 is embedded; it does not require a microprocessor either on or off the memory centric device.

2. Address Decode Operations.

A microprocessor can be used to provide the I/O driver or starting address to bring the micro-code into the MCC. The microprocessor addresses its memory or program to obtain address information or a predetermined bit pattern to send via Z to the MCC. The MCC includes or has access to address decoding logic (not shown) where the received address or bit pattern is decoded and can initiate any one of the following operations, each implied by a predetermined address or bit pattern:

1. Load microcode from external memory into the engine 16 via MCC 30;

2. Initiate an actual memory centric library operation such as calculate FFT, FIR, convolution, MPEG encode/decode, etc. Includes initiating a subroutine for re-ordering locations for certain algorithm requirements, as described in the MCC application identified above.

3. Initiate an I/O operation between the microprocessor and the engine. The decoding used to control or "trigger" operation of the MCC can be

implemented in several ways:

a. Hard-wired logic; very fast decode, simple to implement.

b. Software (microcode) programmable. Allows microcode to determine what address or bit pattern decodes to what operation; highly flexible, a little more complex, slower to decode.

c. Combination — e.g. 4 or 8 hard-wired decodes, together with others programmable. Notice, at least a few hard wired addresses are useful for downloading or "bootstrapping" more microcode, for power-up reset and other special operations.

The address decoding logic can be implemented in the MCC and or in the BIC. Importantly, the operations required to run the memory-centric engine do not require the use of any extended instructions of the processor. In other words, the interface is adaptable through microcode as described above to comply with virtually any existing bus or processor interface standard. The interface apparatus and methods disclosed here are applicable independently of whether a processor coexists on the same chip as the memory-centric engine or is external to it. This allows wide applicability of the new MC architecture *without* having to custom modify exiting processors, bus architectures, cores, etc. Accordingly, one architecture -- one chip design -- can be used for many applications while providing performance at least on a par with more expensive full-custom designs and other programmable solutions.

3. Operation Example using Status Block

Following is an example of a sequence of operations of the architecture:

STEP 1. Address 21x710 Initiates loading of microcode into chip under processor control

STEP 2. Processor Provides micro-code data to MC engine as a form of DMA transfer to memory;

STEP 3. Address 21x777 This address decodes in the MCC to initiate

an internal execution operation in the engine such as FFT, FIR, etc.

STEP 4. Address 23x020 This address is decoded by the MCC — causing it to evaluate the **status block** resident within the MC engine.

The status word or block can be part of the DRAM array, or a dedicated register in the MCC or in the BIC. Status information can include, for example, read overflows, library operations completed (e.g. FFT); or re-ordering data completed, etc. Also, with regard to BIST (built-in self test), the test results can be formed in the status block and then read out via a standard interface. Built-in self test is further described in the MCC application.

STEP 5. Status word or words are then sent directly back to the processor; via the memory sending back data words to a series of consecutive (or ordered) addresses supplied by the processor after the address 23x020 is asserted as above. Thus, for example, overflows, status of execution (extent completed), operations completed and waiting for I/O command are examples of status bits or data words that can be returned to the processor.

One can observe from this description several advantages. For example, a processor can load and start the microcontroller executing a selected function without any special or custom interface signals or control or data paths. Any standard bus or memory interface can serve as the interface to the MC engine. Moreover, the status block or word can be used to provide status information to the processor whenever desired, thus allowing the processor to poll the location. In other applications where processor interrupts are implemented, they could be used along with or as alternative to the polled status block. The status block is highly flexible, and can provide as much (or as little), information as required. For example, it could be used to provide detailed information about ongoing executions. It can also provide self-test results.

To summarize this aspect of the invention, the interface methodology provides for loading microcode, initializing various operations, retrieving status information, etc. all *without* changing standard interface hardware, signals and

while using only standard (native) processor instructions. In addition, the bus interface controller and associated buffer memory accommodate high-speed interfaces as required, while the MCC and DRAM array generally operate at relatively low speed, thereby reducing power consumption in the chip.

Having described and illustrated the principles of the invention in a preferred embodiment thereof, it should be apparent that the invention can be modified in arrangement and detail without departing from such principles. I claim all modifications and variation coming within the spirit and scope of the following claims

I CLAIM:

1. A method of interfacing a processor bus to a computation engine having a microprogrammable memory-centric controller and an array of memory, the method comprising the steps of:

providing a predetermined series of microcode instructions for execution by the MCC;

selecting a start address within the series of microcode instructions for carrying out a corresponding operation; and

executing the series of microcode instructions in the MCC beginning at the selected start address so as to carry out the corresponding operation in the engine.

2. A method according to claim 1 wherein said providing the series of microcode instructions comprises storing the microcode instructions in a non-volatile memory accessible to the MCC.

3. A method according to claim 1 wherein said providing the series of microcode instructions comprises storing the microcode instructions in a non-volatile external memory accessible to the MCC.

4. A method according to claim 1 wherein said providing the series of microcode instructions comprises downloading the microcode instructions under processor control to a microcode storage memory accessible to the MCC.

5. A method according to claim 1 wherein said providing the series of microcode instructions comprises downloading the microcode instructions under processor control into the array of memory in the computation engine.

6. A method according to claim 5 wherein said downloading the microcode instructions includes asserting a predetermined address; decoding the predetermined address in the MCC; and in response to the decoding step, configuring the engine for storing the microcode instructions under processor control into the array of memory.

7. A method according to claim 6 wherein said configuring step includes coupling an address provided by the processor to the memory array so as to address the memory array under processor control.

8. A method according to claim 1 wherein said selecting a start address in the microcode includes receiving a predetermined address from the processor and decoding the said address so as to identify a corresponding start address in the microcode, thereby selecting the corresponding operation without requiring a custom or dedicated interface to the engine.

9. A method according to claim 1 wherein:

the computing engine includes an SRAM buffer memory and the memory array comprises an array of DRAM memory;

the corresponding operation is a data write operation into the computing engine; and

said executing step includes storing data from the bus into the SRAM buffer memory and then transferring the stored data from the buffer memory into the DRAM array.

10. A method according to claim 9 wherein said transferring the stored data to the DRAM array includes writing a series of data words into a latch and then writing the contents of the latch into the DRAM array in a single write operation so as to improve matching access time of the SRAM buffer memory to access time of the DRAM array.

11. A method of interfacing a processor bus to a computation engine having a microprogrammable memory-centric controller and an array of memory, the method comprising the steps of:

providing a predetermined series of microcode instructions for execution by the MCC;

selecting a start address within the series of microcode instructions for carrying out a corresponding operation; and

executing the series of microcode instructions in the MCC beginning at the selected start address so as to carry out the corresponding operation in the engine; wherein said selecting step includes decoding an address asserted by the processor.

12. A method according to claim 11 wherein said decoding step includes providing hard-wired decoding logic in the MCC arranged so that at least one selected address automatically decodes so as to initiate a predetermined operation.

13. A method according to claim 12 wherein said operation comprises downloading microcode into the MCC.

14. A method according to claim 11 wherein said decoding step includes microprogramming the MCC so as to enable it to decode the selected address.

15. A method according to claim 14 wherein said microprogramming step includes providing a lookup table so as to enable the MCC to translate the asserted address into a microcode start address.

16. A memory-centric computing engine comprising:

an array of DRAM memory;

an execution unit coupled to the DRAM array for executing selected operations on data stored in the DRAM array;

a memory-centric controller for controlling and addressing the DRAM array; and

a bus interface controller for interfacing the engine to an external bus.

17. A memory-centric computing engine according to claim 16 and further comprising a buffer memory under control of the bus interface controller for buffering data transfers between the internal data bus and the external bus.

18. A memory-centric computing engine according to claim 17 wherein the buffer memory comprises SRAM memory cells.

19. A memory-centric computing engine according to claim 16 further comprising logic coupled to the DRAM array data port for improving access time matching between the external bus and the DRAM array.

20. A memory-centric computing engine according to claim 19 wherein said logic includes a latch and a multiplexer for writing multiple data words into the latch.

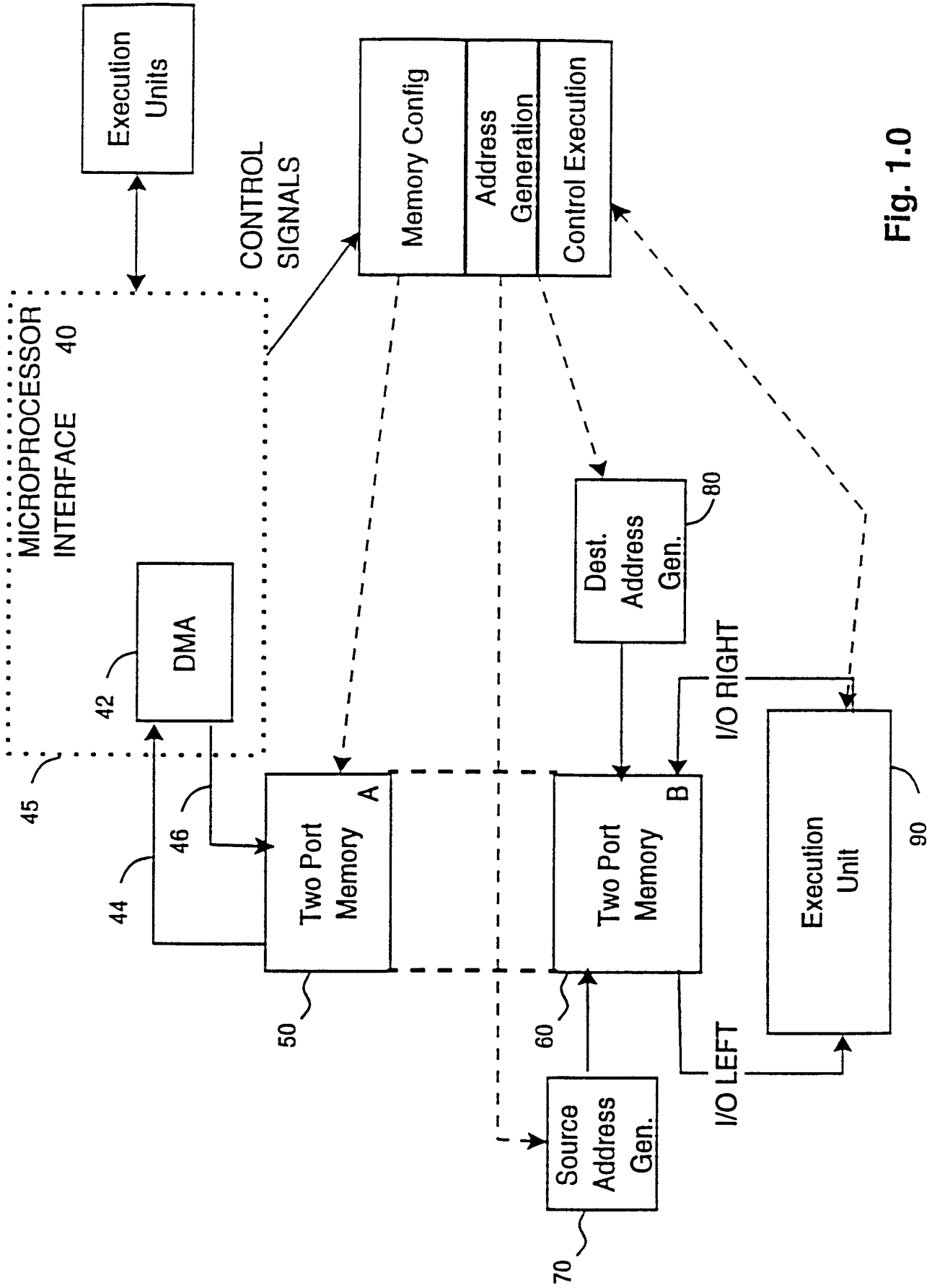

Execution Units
CONTROL SIGNALS
Memory Config
Address Generation
Control Execution
MICROPROCESSOR INTERFACE 40
DMA
42
45
46
44
Two Port Memory A
50
Two Port Memory B
60
Dest. Address Gen.
80
I/O RIGHT
I/O LEFT
Execution Unit
90
Source Address Gen.
70

Fig. 1.0

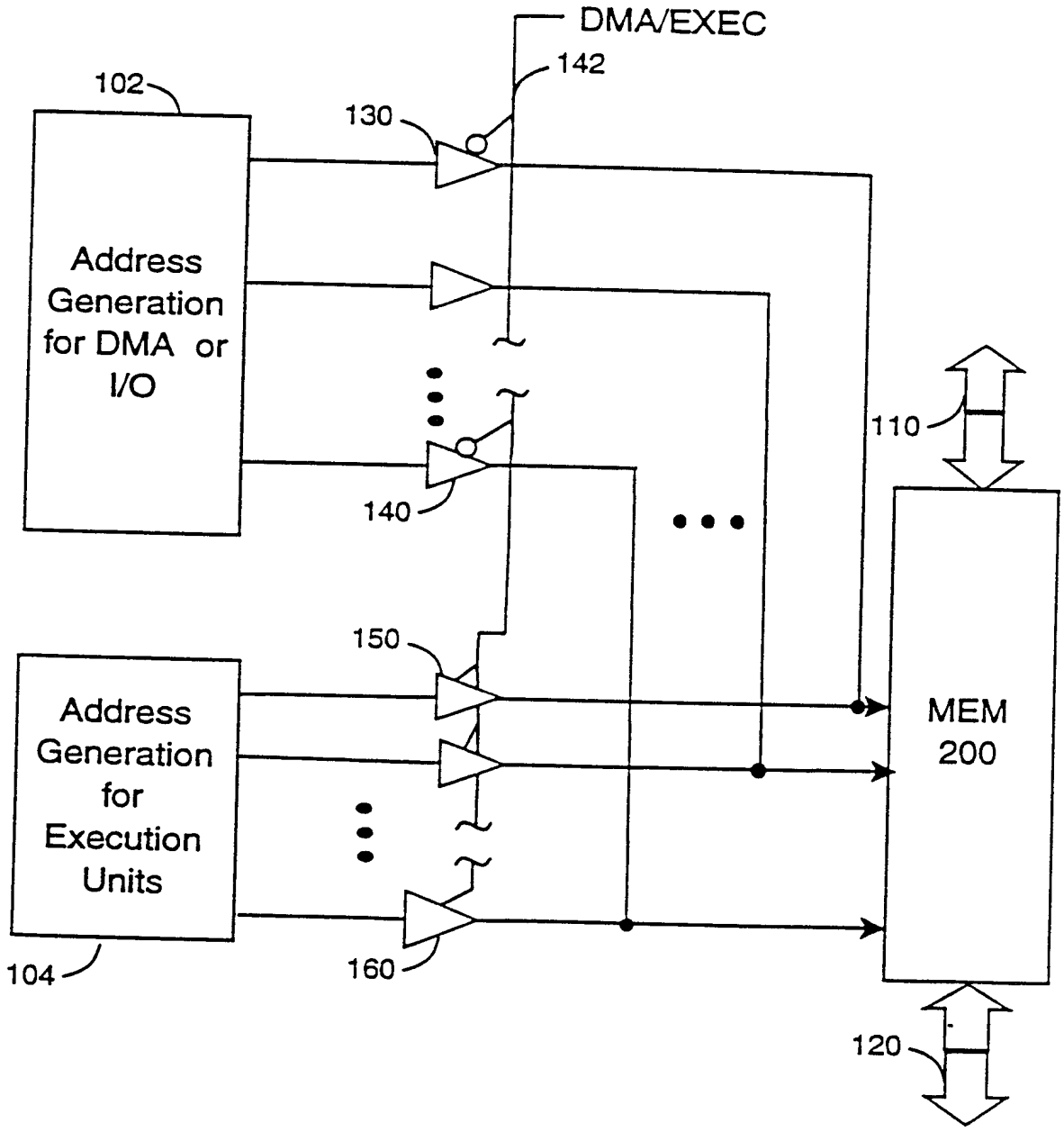

DMA/EXEC
142
102
130
Address Generation for DMA or I/O
110
140
150
Address Generation for Execution Units
MEM 200
104
160
120

FIG. 2

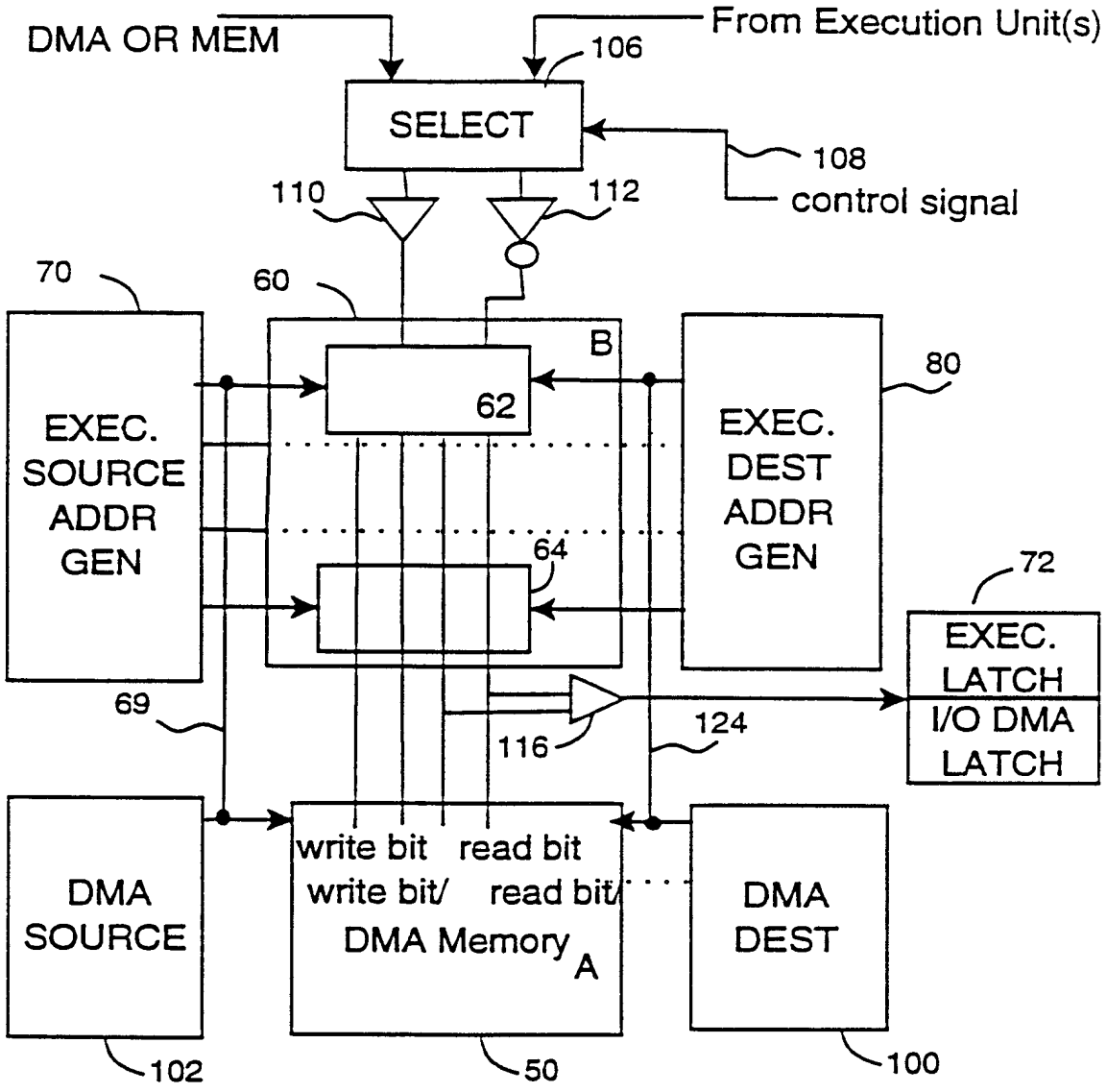

DMA OR MEM
From Execution Unit(s)
106
SELECT
108
control signal
110
112
70
60
B
62
80
EXEC. SOURCE ADDR GEN
EXEC. DEST ADDR GEN
64
72
EXEC. LATCH
I/O DMA LATCH
69
116
124
DMA SOURCE
write bit
read bit
write bit/
read bit/
DMA Memory A
DMA DEST
102
50
100

Fig. 3.0

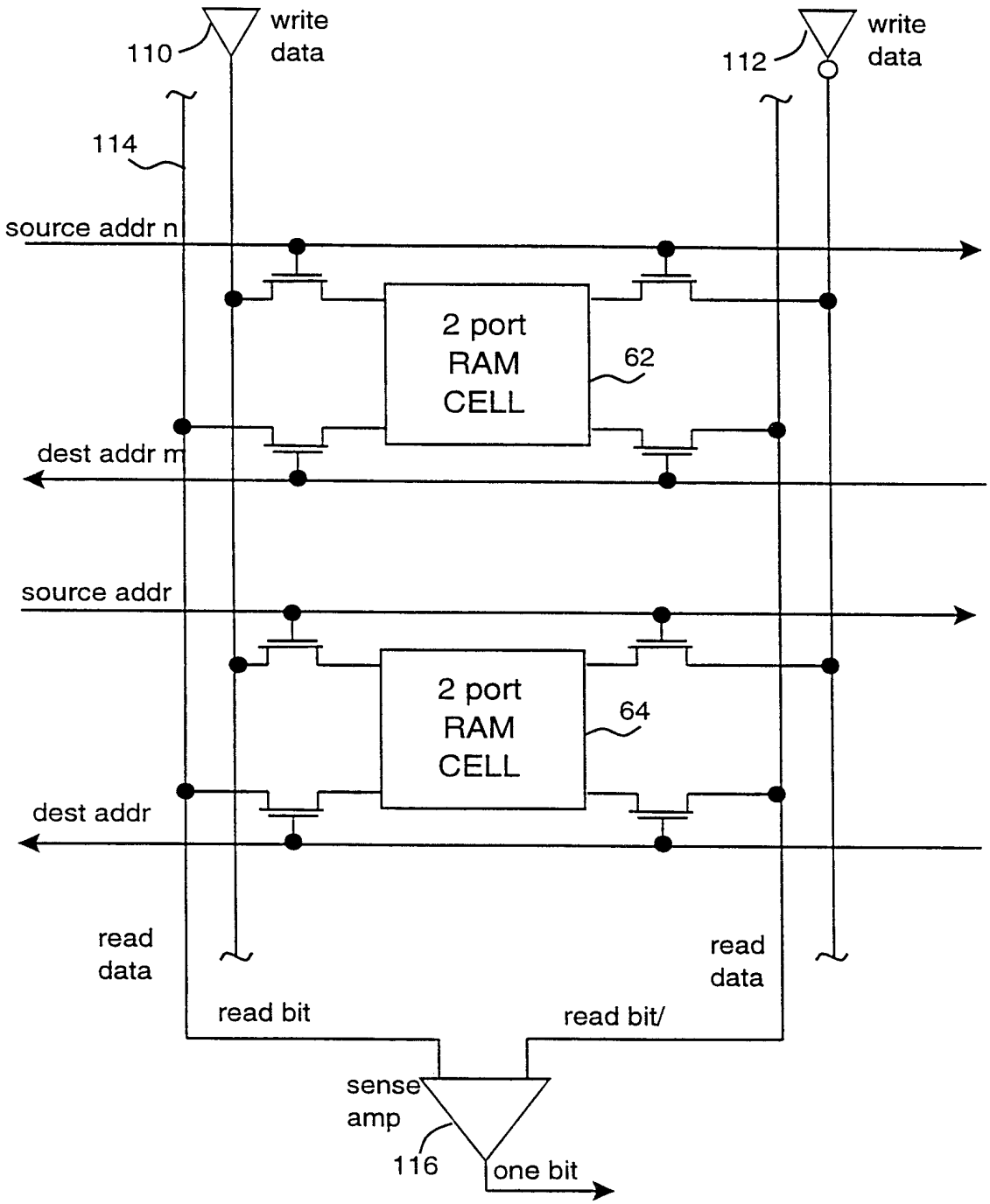

write
data
110
write
data
112
114
source addr n
2 port
RAM
CELL
62
dest addr m
source addr
2 port
RAM
CELL
64
dest addr
read
data
read
data
read bit
read bit/
sense
amp
116
one bit

FIG. 4

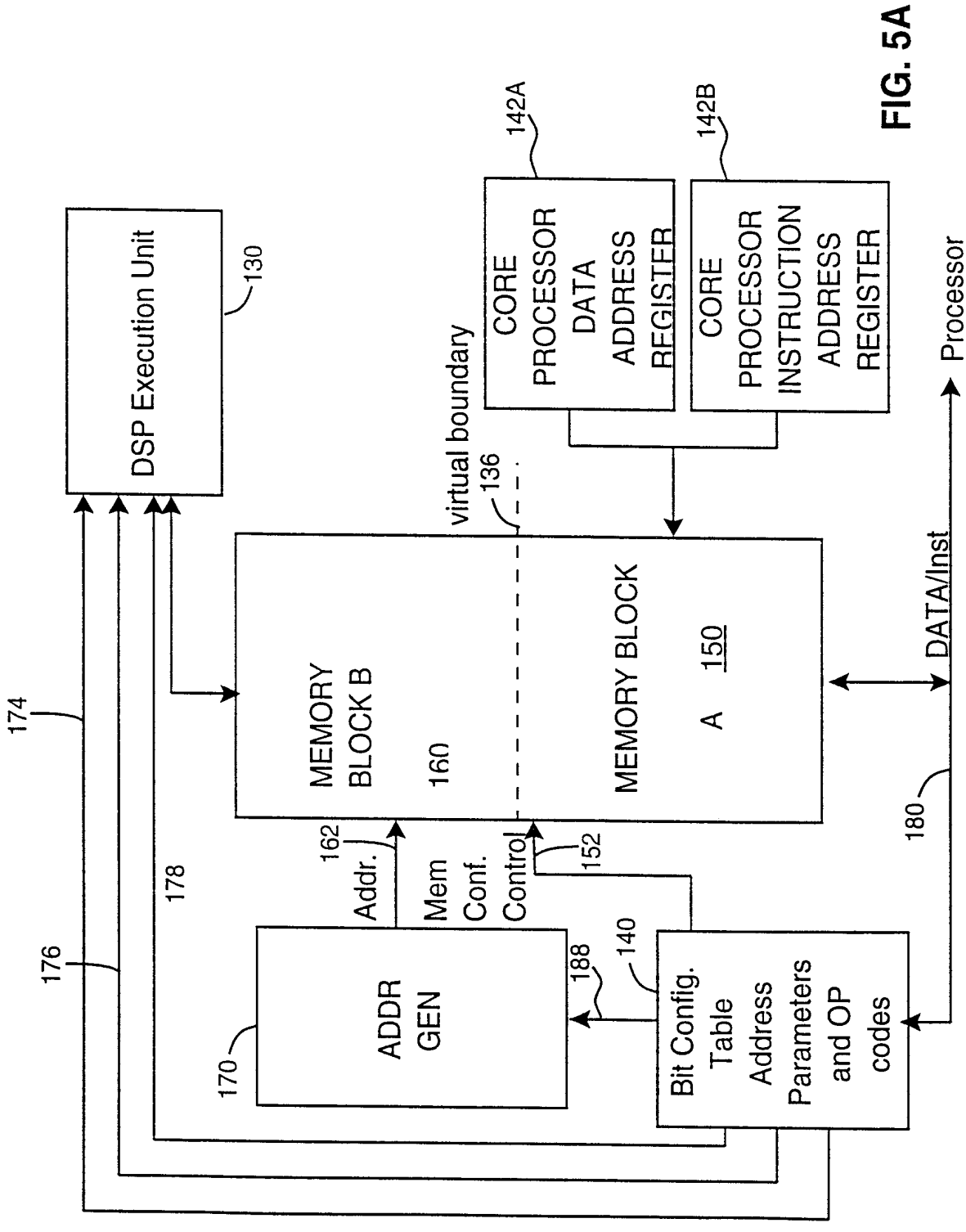

DSP Execution Unit
130
142A
142B
CORE PROCESSOR DATA ADDRESS REGISTER
CORE PROCESSOR INSTRUCTION ADDRESS REGISTER
Processor
virtual boundary
136
MEMORY BLOCK B 160
MEMORY BLOCK A 150
DATA/Inst
174
176
178
180
162
Addr.
Mem Conf. Control
152
140
188
ADDR GEN
170
Bit Config. Table Address Parameters and OP codes

FIG. 5A

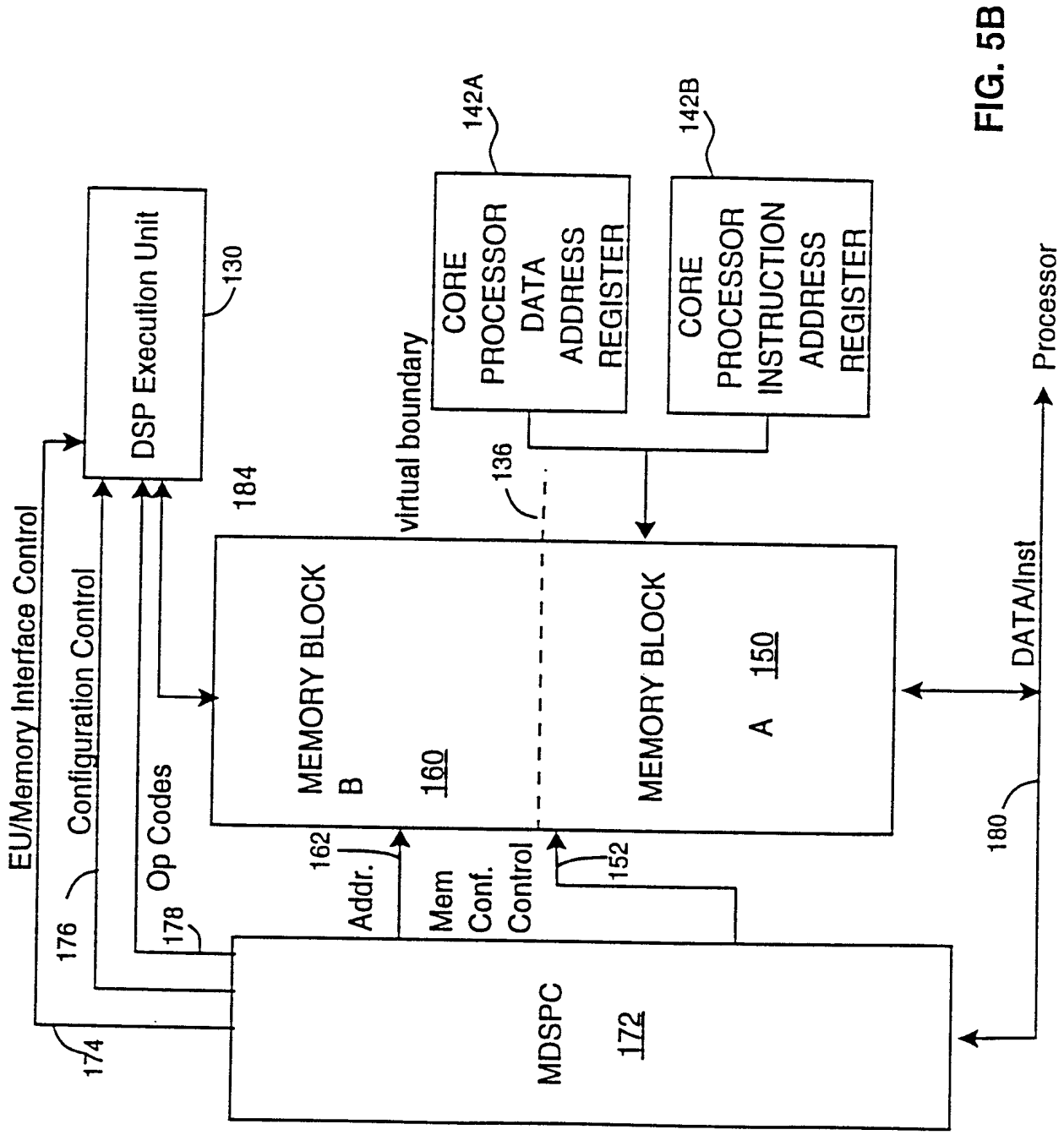

EU/Memory Interface Control
Configuration Control
Op Codes
DSP Execution Unit
130
184
virtual boundary
136
142A
CORE PROCESSOR DATA ADDRESS REGISTER
142B
CORE PROCESSOR INSTRUCTION ADDRESS REGISTER
Processor
DATA/Inst
MEMORY BLOCK B 160
MEMORY BLOCK A 150
162
Addr.
Mem Conf. Control
152
176
178
174
180
MDSPC 172

FIG. 5B

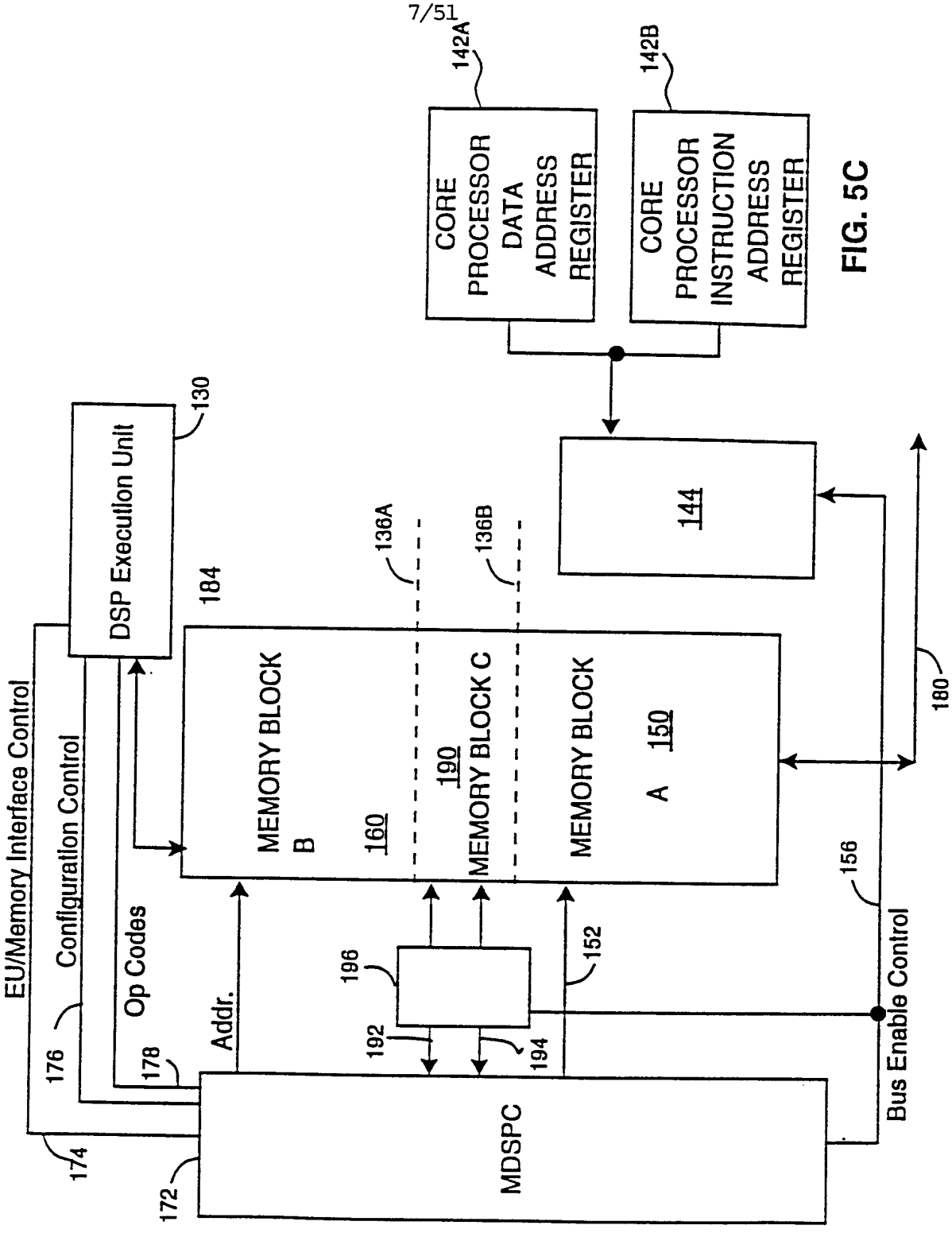

142A
142B
CORE PROCESSOR DATA ADDRESS REGISTER
CORE PROCESSOR INSTRUCTION ADDRESS REGISTER
130
DSP Execution Unit
144
184
136A
136B
MEMORY BLOCK B
160
190
MEMORY BLOCK C
MEMORY BLOCK A
150
180
EU/Memory Interface Control
Configuration Control
Op Codes
Addr.
176
178
196
192
194
152
156
Bus Enable Control
174
172
MDSPC

FIG. 5C

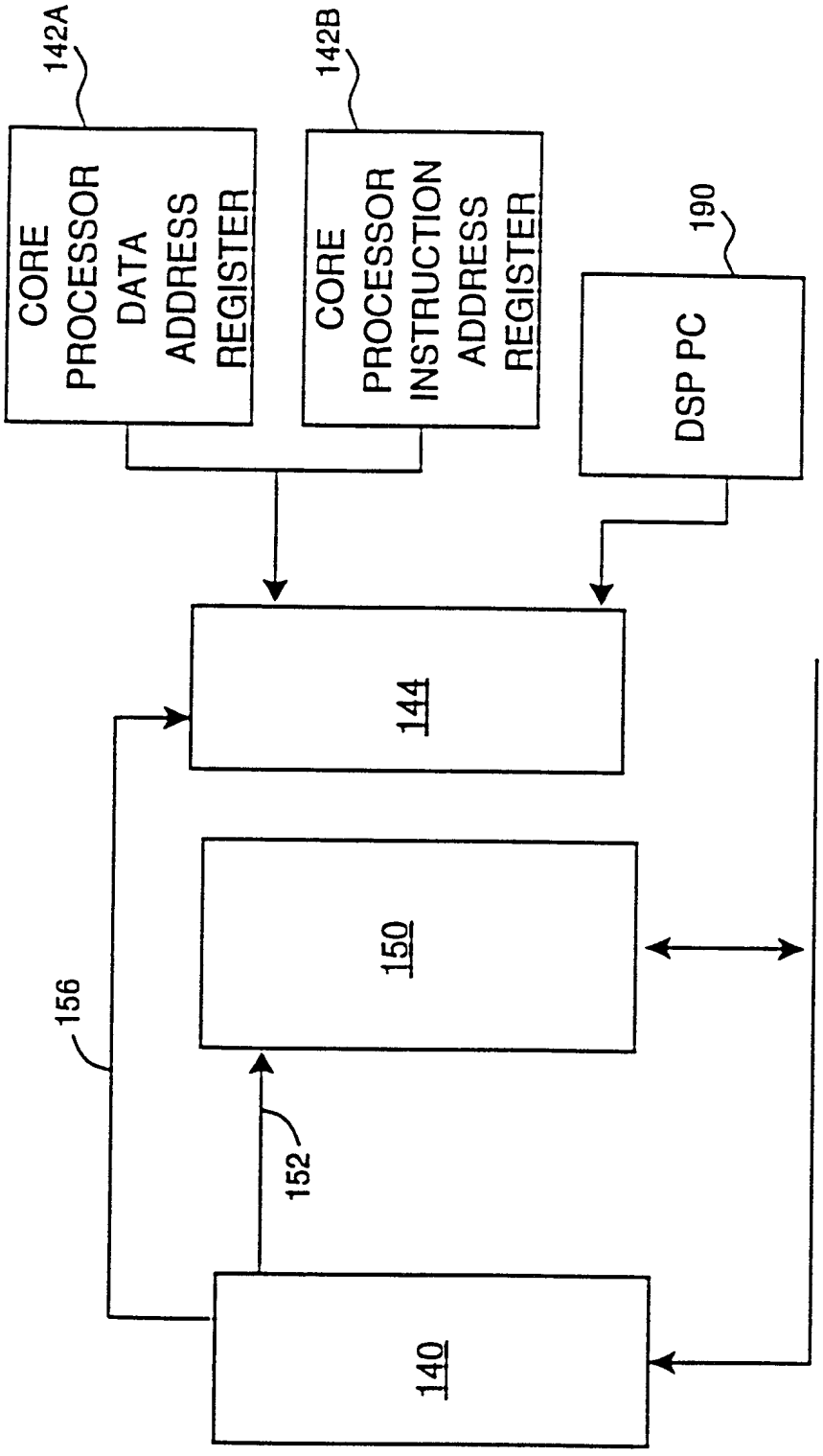

142A
CORE PROCESSOR DATA ADDRESS REGISTER
142B
CORE PROCESSOR INSTRUCTION ADDRESS REGISTER
190
DSP PC
144
150
156
152
140

FIG. 6A

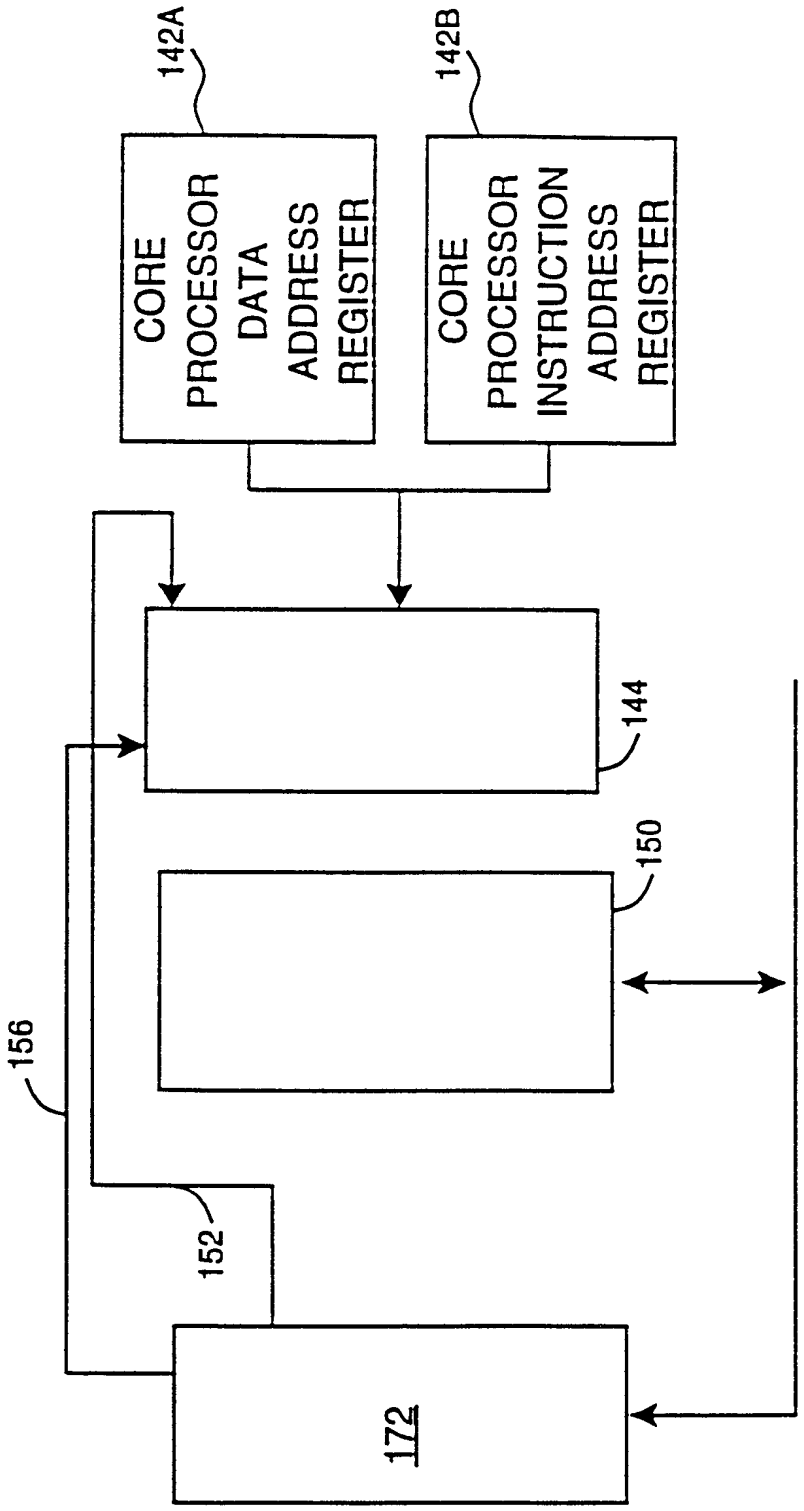

142A
142B
CORE PROCESSOR DATA ADDRESS REGISTER
CORE PROCESSOR INSTRUCTION ADDRESS REGISTER
144
150
156
152
172

FIG. 6B

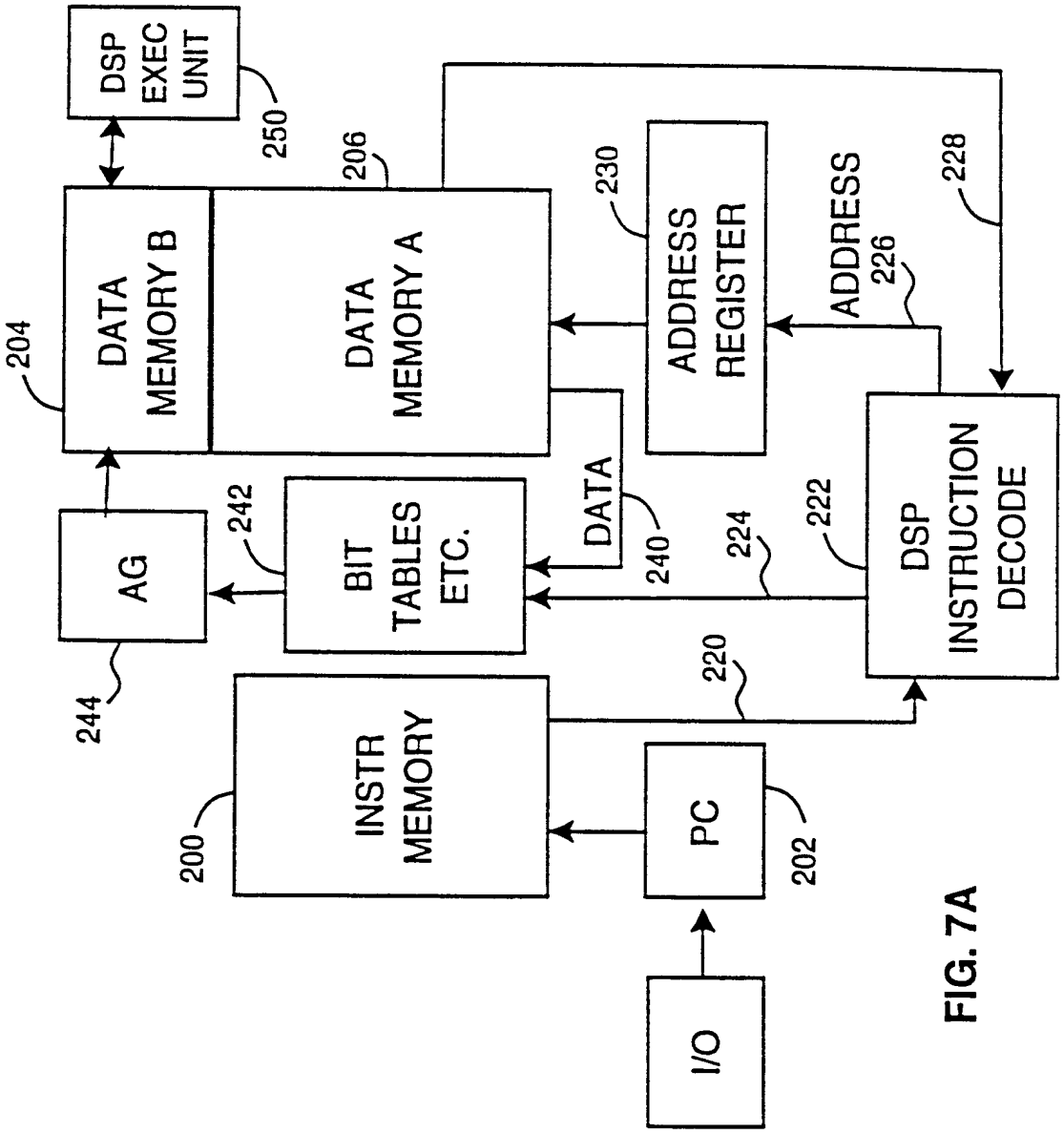

DSP EXEC UNIT
250
206
230
228
DATA MEMORY B
DATA MEMORY A
ADDRESS REGISTER
ADDRESS
226
204
242
BIT TABLES ETC.
DATA
240
224
222
DSP INSTRUCTION DECODE
AG
244
220
INSTR MEMORY
200
PC
202
I/O

FIG. 7A

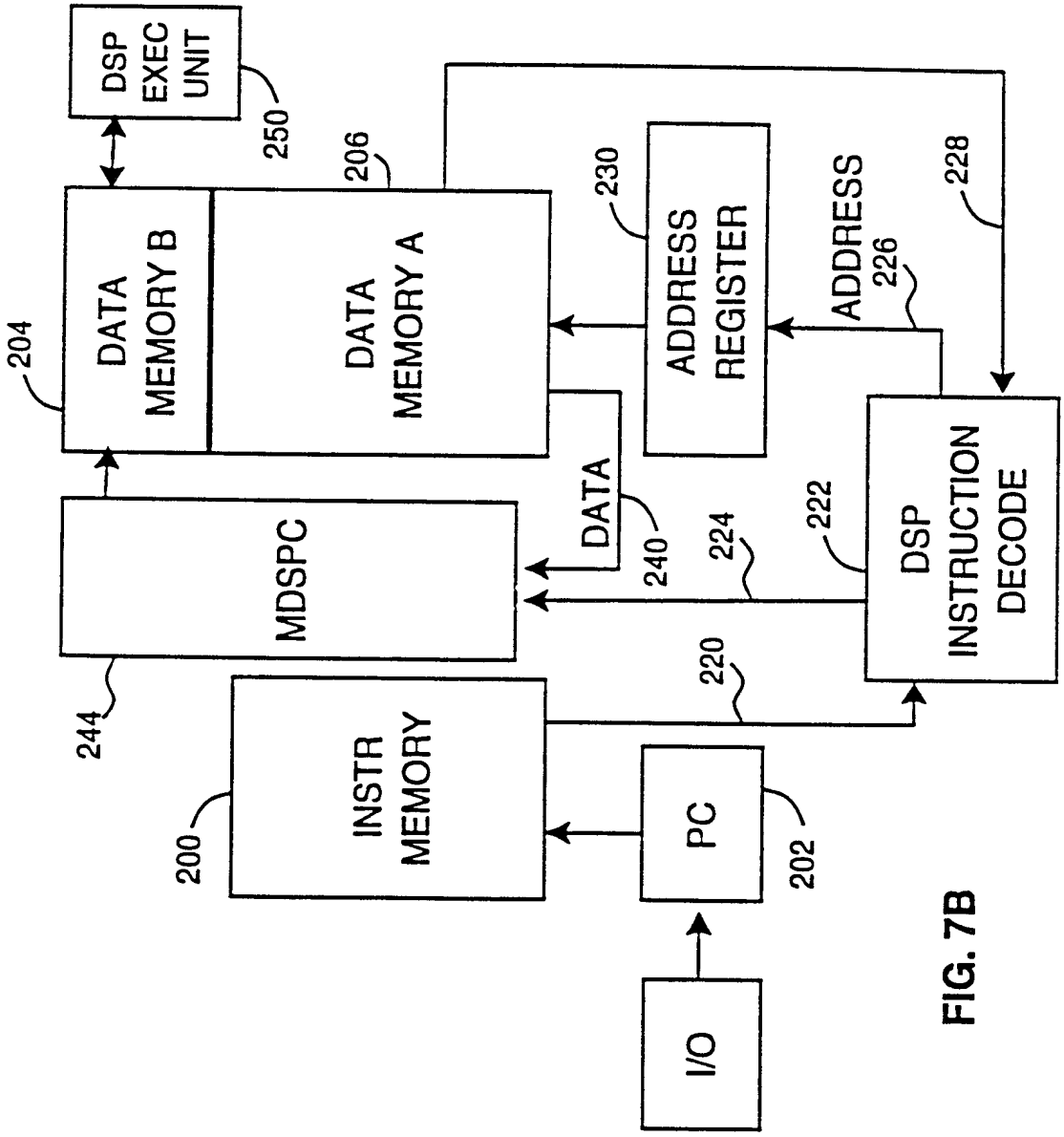

DSP EXEC UNIT
250
206
230
228
DATA MEMORY B
DATA MEMORY A
ADDRESS REGISTER
ADDRESS
226
204
DATA
240
224
222
MDSPC
DSP INSTRUCTION DECODE
220
244
INSTR MEMORY
200
PC
202
I/O

FIG. 7B

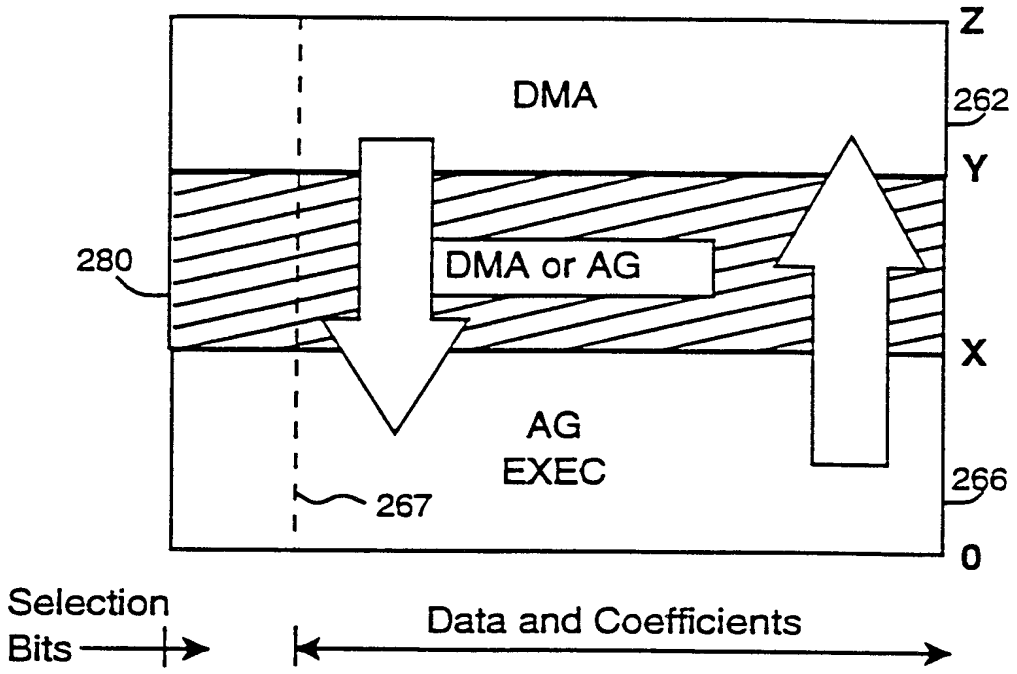

Z
DMA
262
Y
280
DMA or AG
X
AG
EXEC
266
267
0
Selection
Bits
Data and Coefficients

FIG. 8

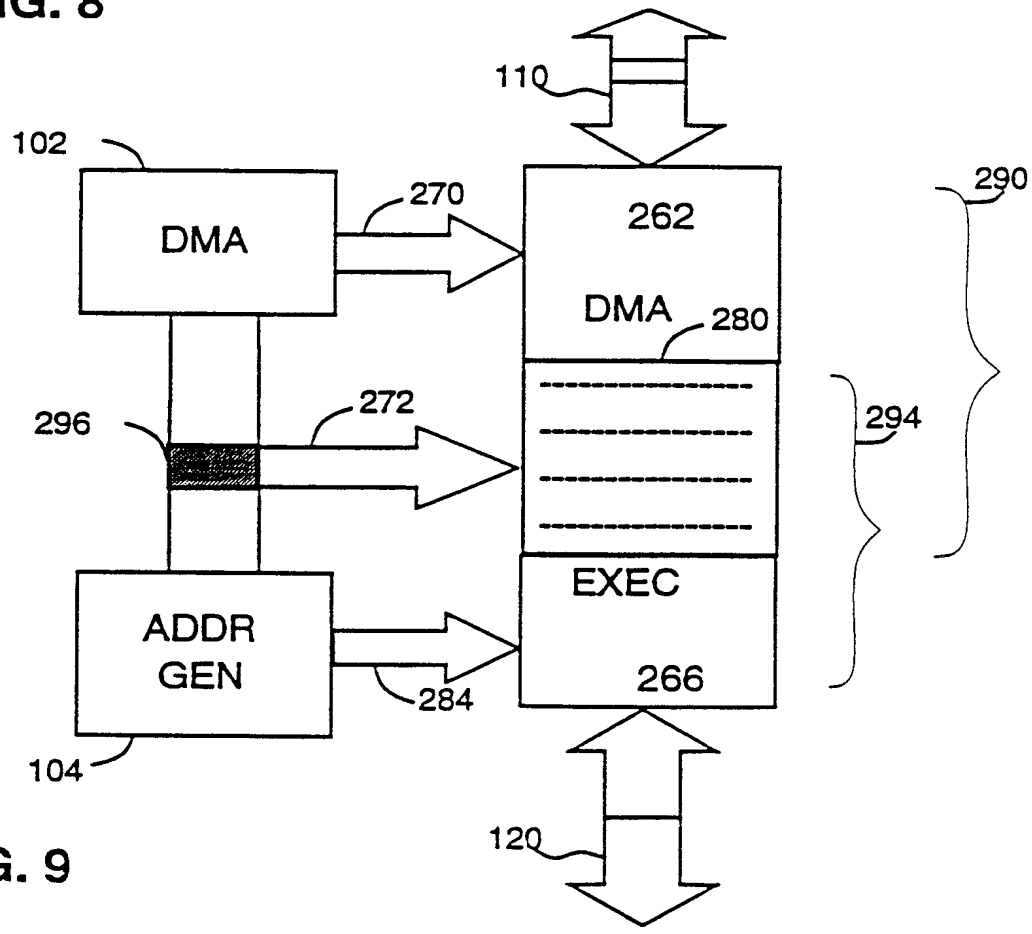

110
102
DMA
270
262
DMA
280
290
296
272
294
EXEC
ADDR
GEN
266
284
104
120

FIG. 9

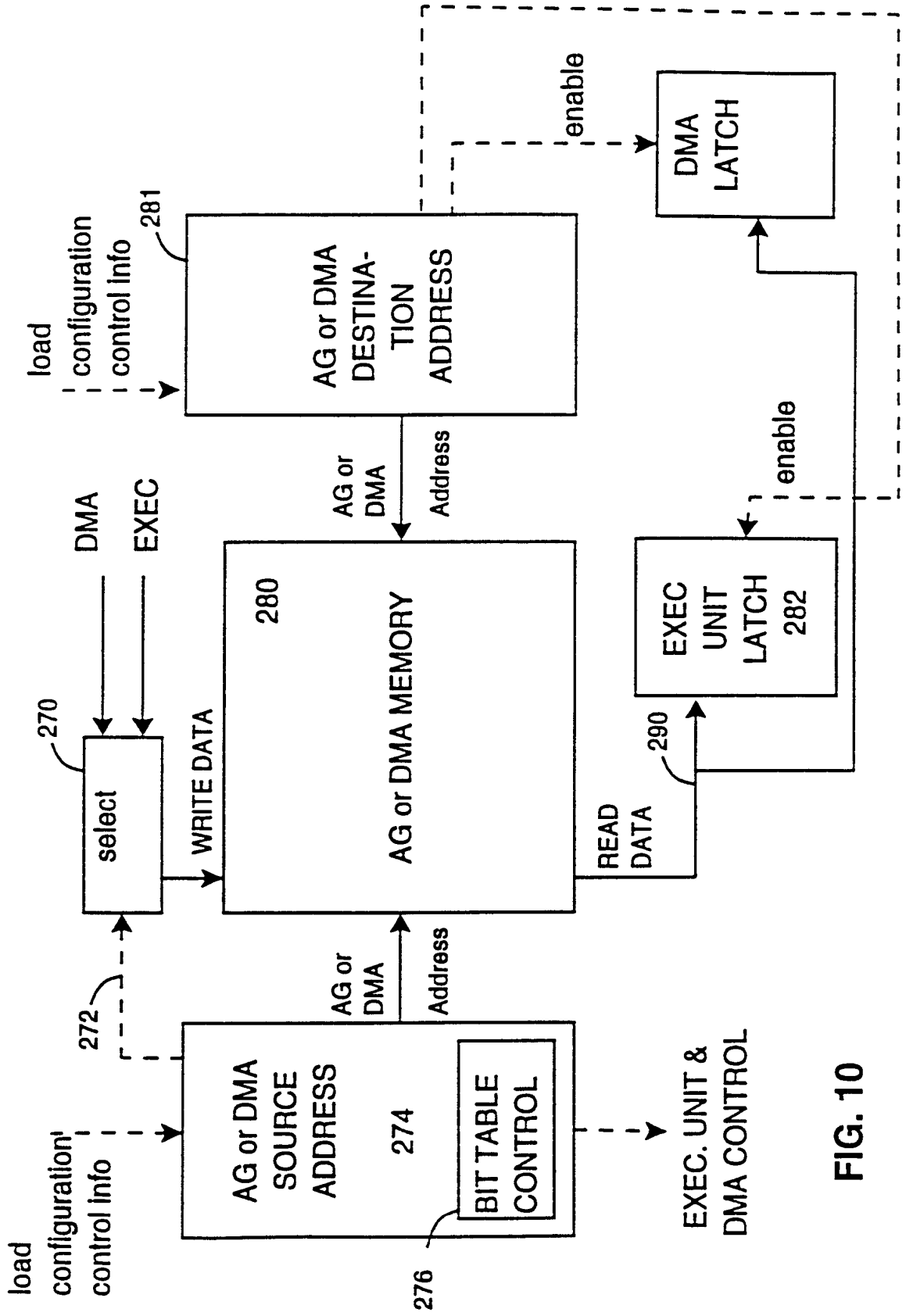

load configuration control info
281
AG or DMA DESTINA-TION ADDRESS
enable
DMA LATCH
DMA
EXEC
AG or DMA
Address
enable
280
AG or DMA MEMORY
EXEC UNIT LATCH 282
270
select
WRITE DATA
290
READ DATA
272
AG or DMA
Address
load configuration control info
AG or DMA SOURCE ADDRESS
274
BIT TABLE CONTROL
276
EXEC. UNIT & DMA CONTROL

FIG. 10

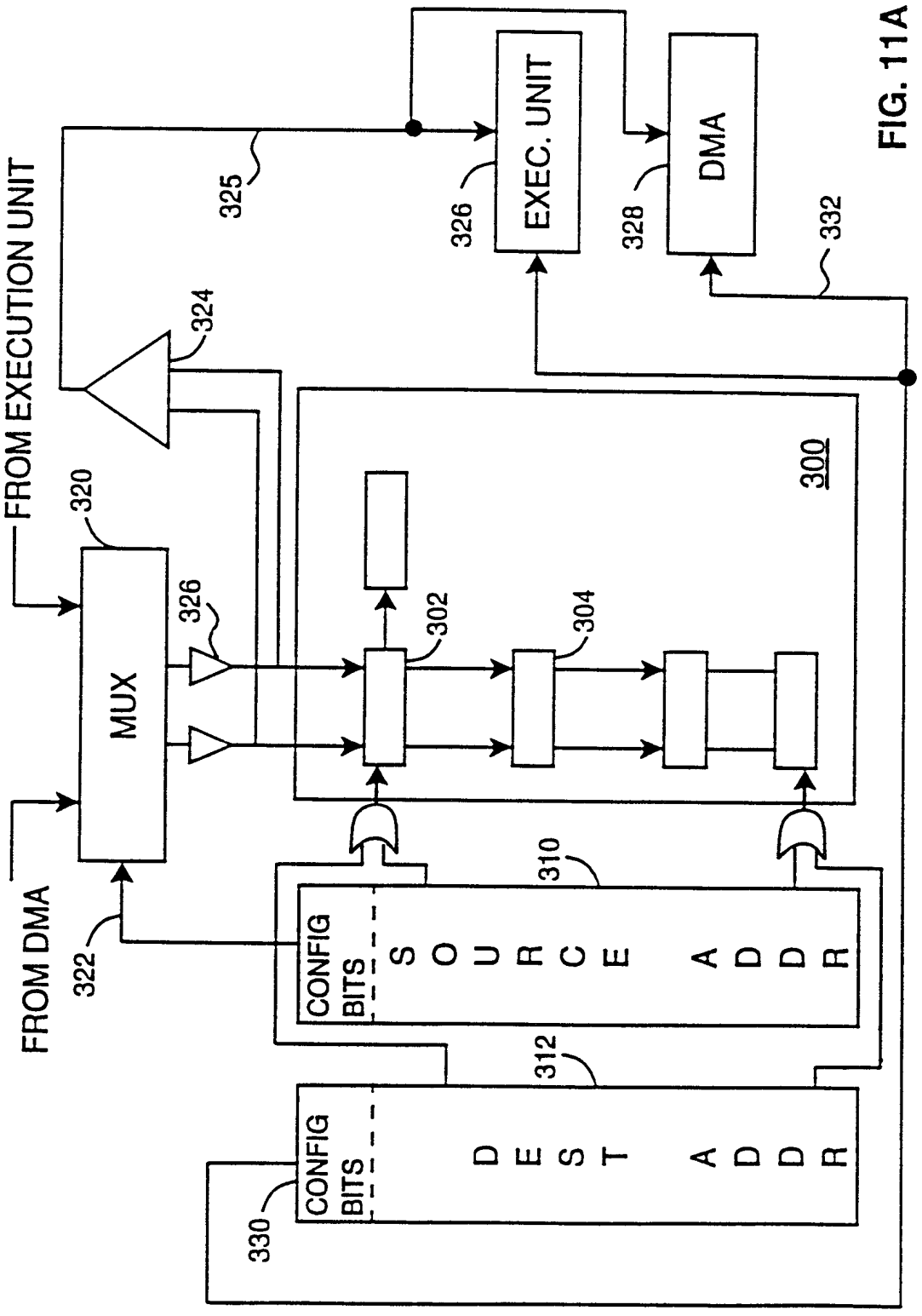

FROM EXECUTION UNIT
FROM DMA
MUX
320
322
326
324
325
326
EXEC. UNIT
328
DMA
332
300
302
304
310
312
330
CONFIG BITS
SOURCE ADDR
CONFIG BITS
DEST ADDR

FIG. 11A

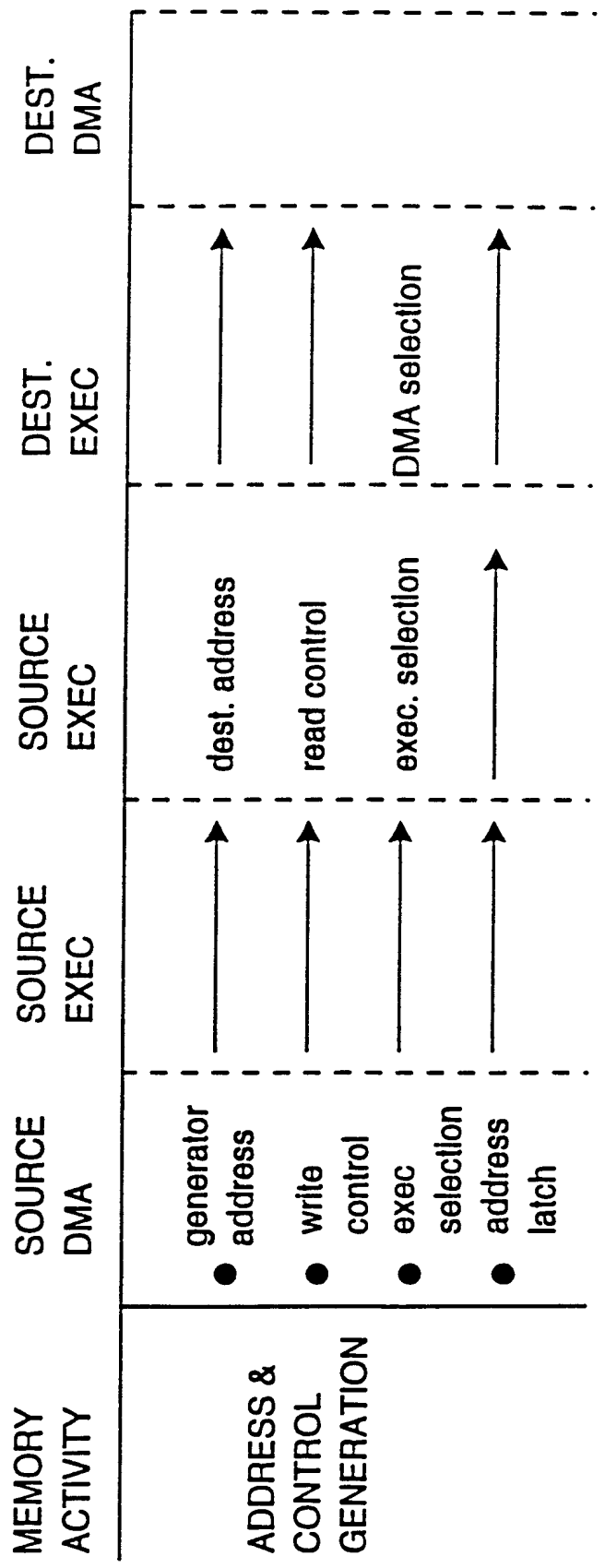

MEMORY ACTIVITY
ADDRESS & CONTROL GENERATION
T1
SOURCE DMA
T2
SOURCE EXEC
T3
SOURCE EXEC
T4
DEST. EXEC
T5
DEST. DMA
generator address
write control
exec selection
address latch
dest. address
read control
exec. selection
DMA selection

FIG. 11B

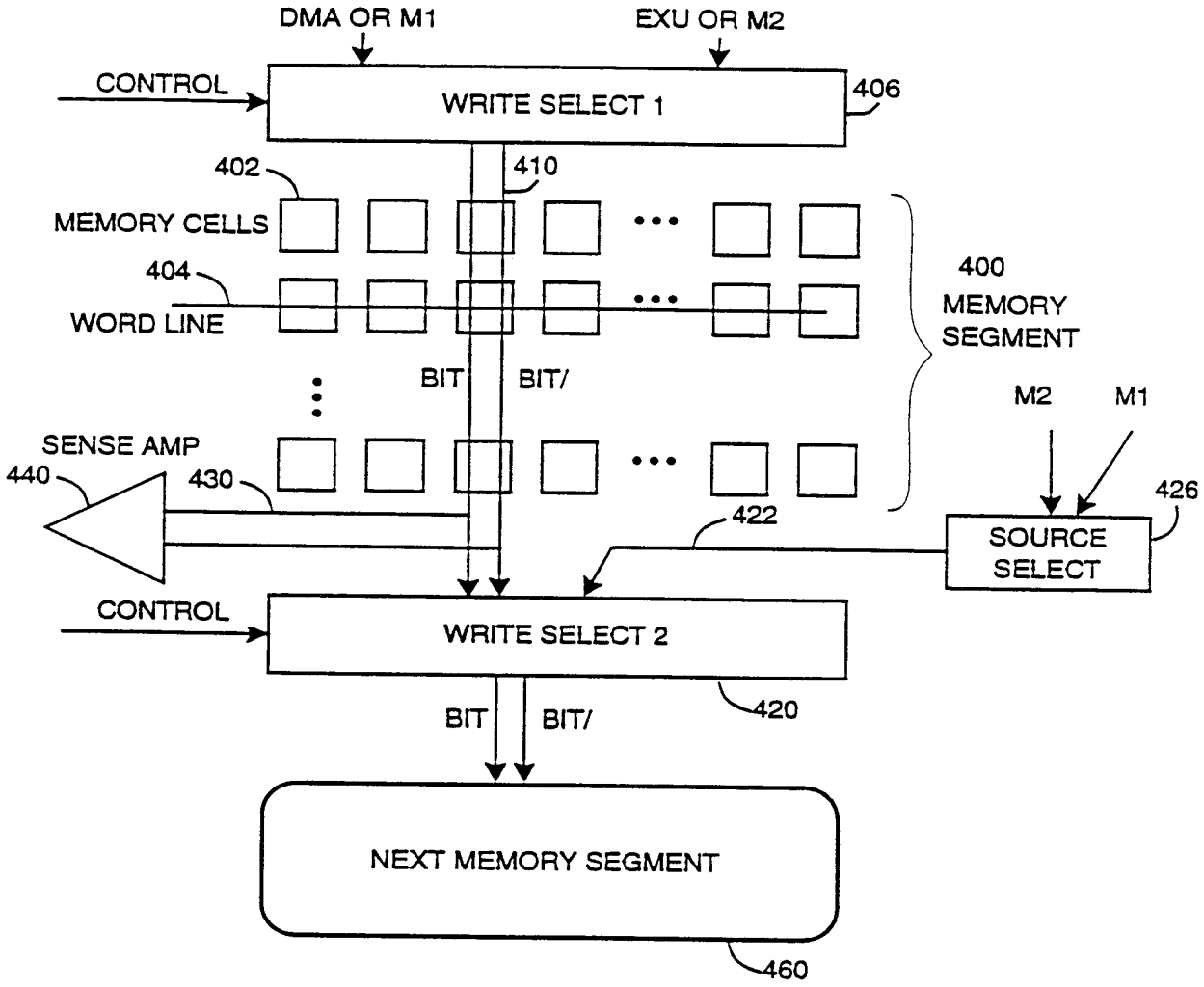

DMA OR M1
EXU OR M2
CONTROL
WRITE SELECT 1
406
402
410
MEMORY CELLS
404
WORD LINE
400
MEMORY
SEGMENT
BIT
BIT/
M2
M1
SENSE AMP
440
430
426
422
SOURCE
SELECT
CONTROL
WRITE SELECT 2
420
BIT
BIT/
NEXT MEMORY SEGMENT
460

FIG. 12

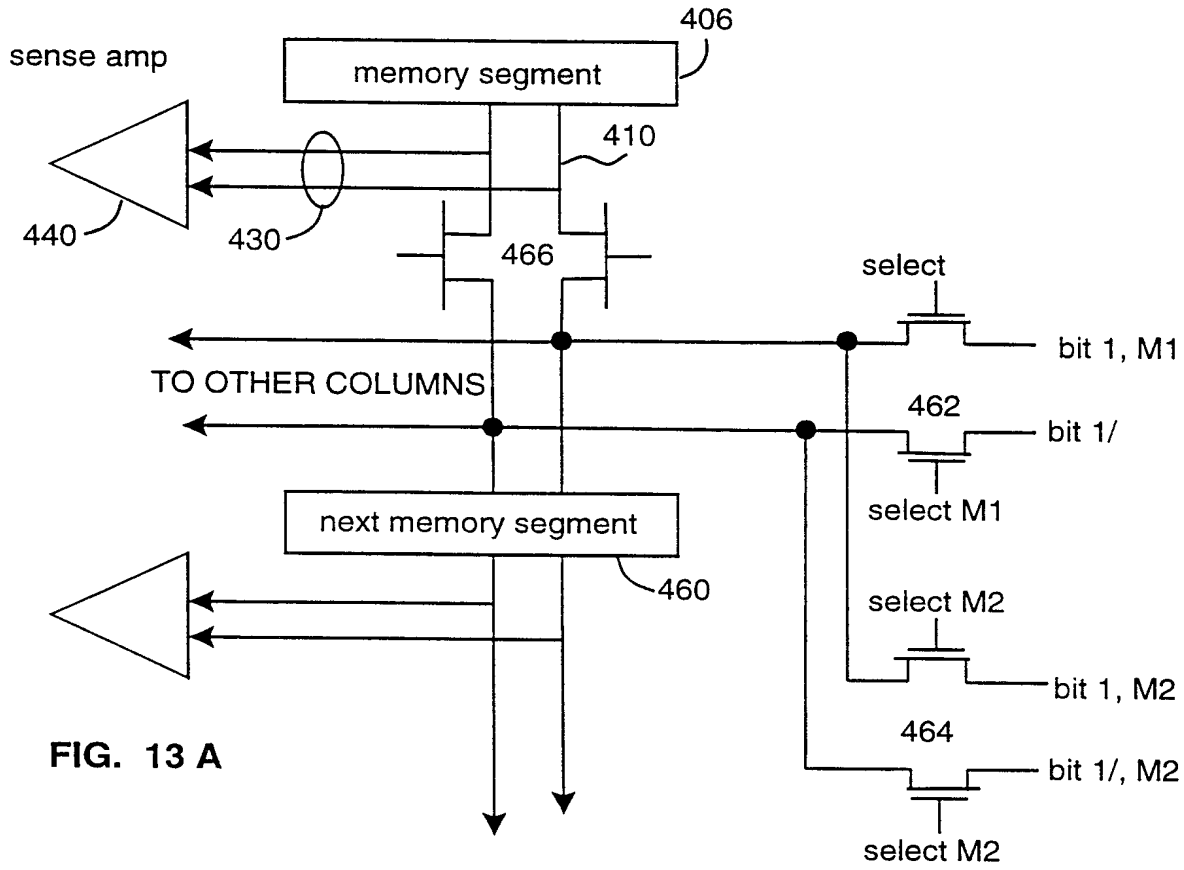


FIG. 13 A

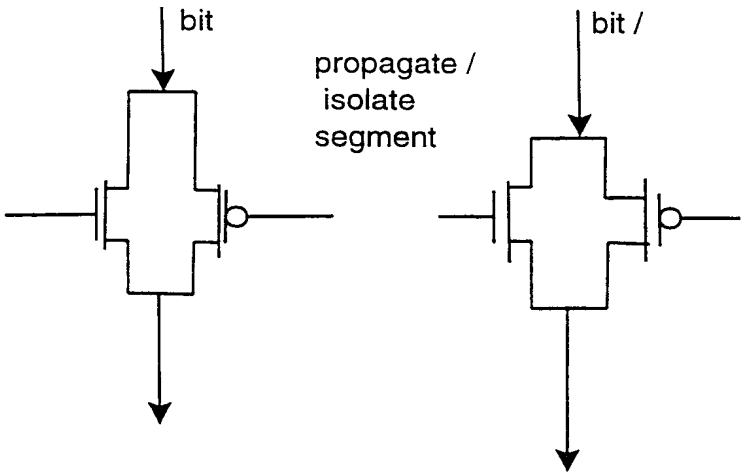


FIG. 13 B

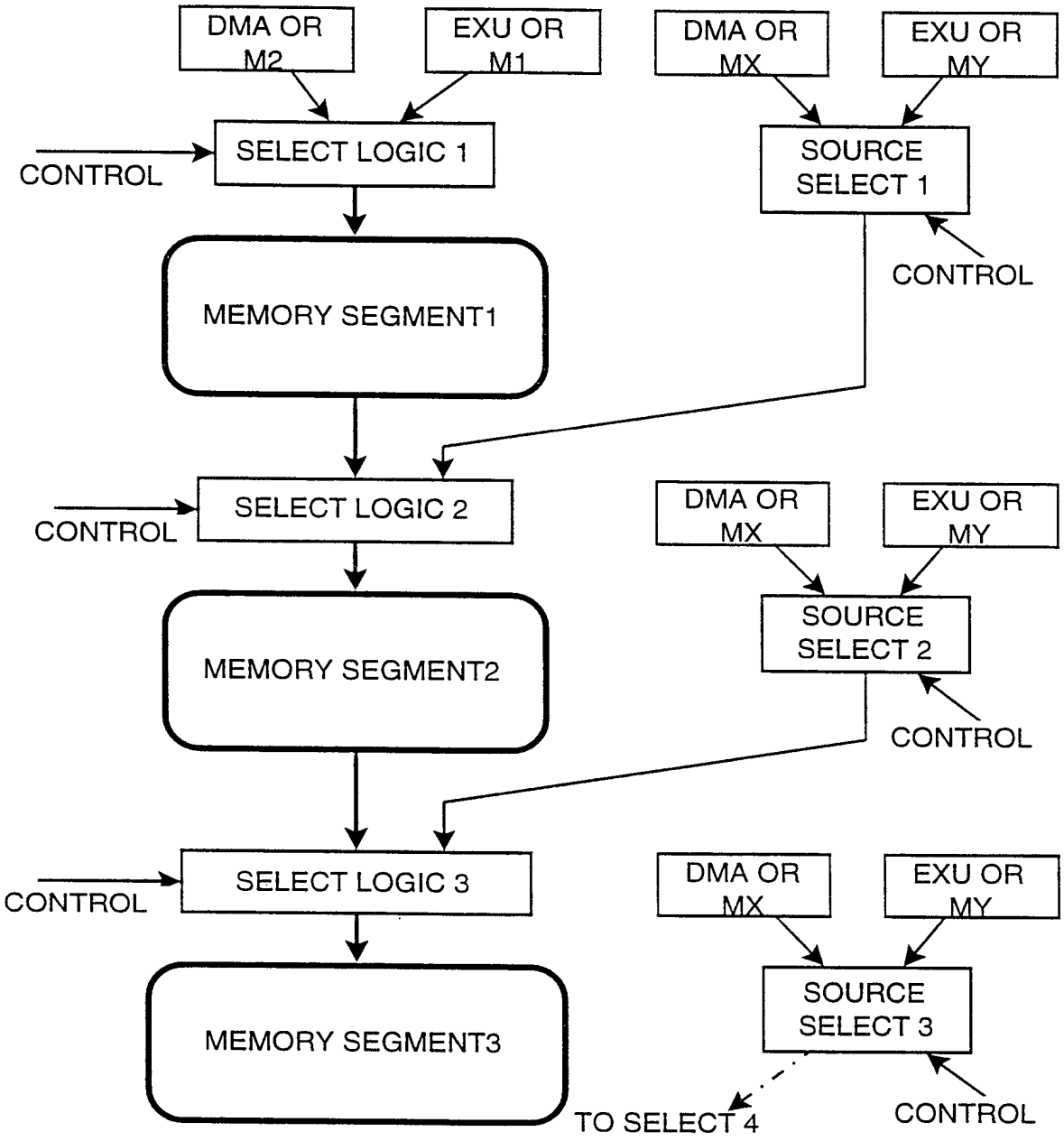

DMA OR M2
EXU OR M1
DMA OR MX
EXU OR MY
SELECT LOGIC 1
CONTROL
SOURCE SELECT 1
CONTROL
MEMORY SEGMENT1
SELECT LOGIC 2
CONTROL
DMA OR MX
EXU OR MY
SOURCE SELECT 2
CONTROL
MEMORY SEGMENT2
SELECT LOGIC 3
CONTROL
DMA OR MX
EXU OR MY
SOURCE SELECT 3
CONTROL
MEMORY SEGMENT3
TO SELECT 4

FIG. 14

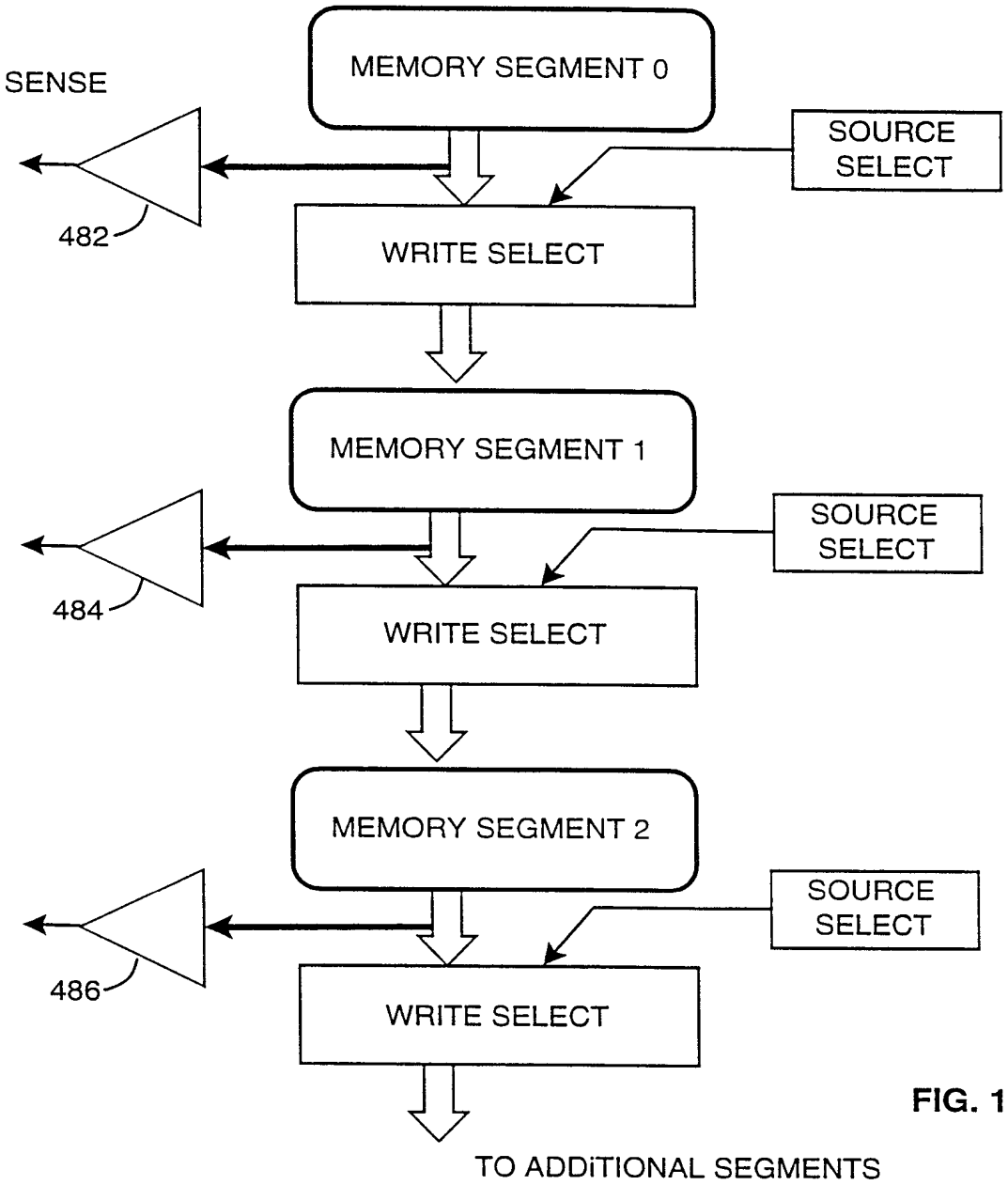

MEMORY SEGMENT 0
SENSE
SOURCE SELECT
482
WRITE SELECT
MEMORY SEGMENT 1
SOURCE SELECT
484
WRITE SELECT
MEMORY SEGMENT 2
SOURCE SELECT
486
WRITE SELECT
TO ADDITIONAL SEGMENTS

FIG. 15

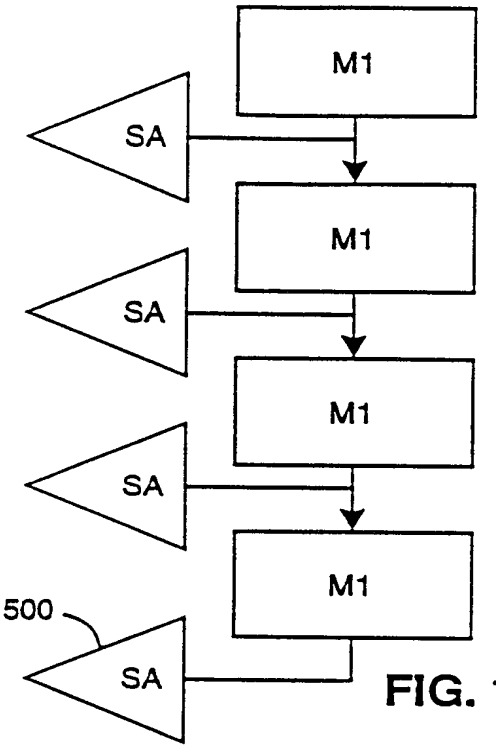


FIG. 16A

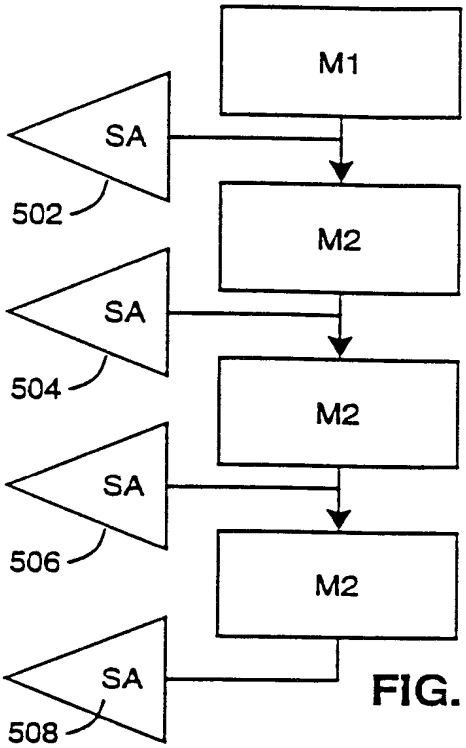


FIG. 16B

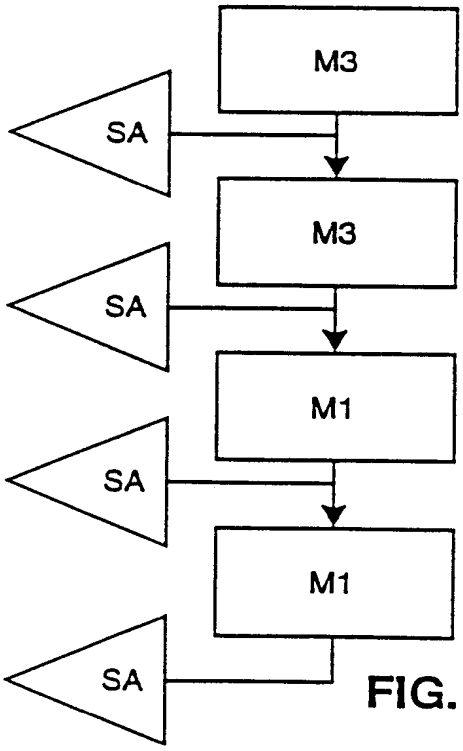


FIG. 16C

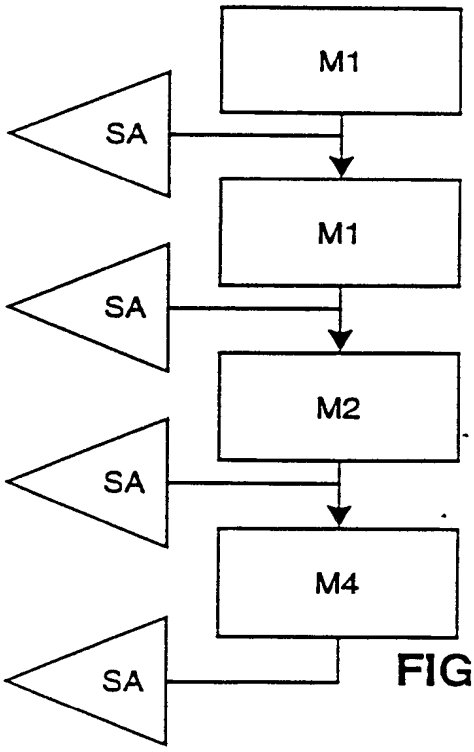


FIG. 16D

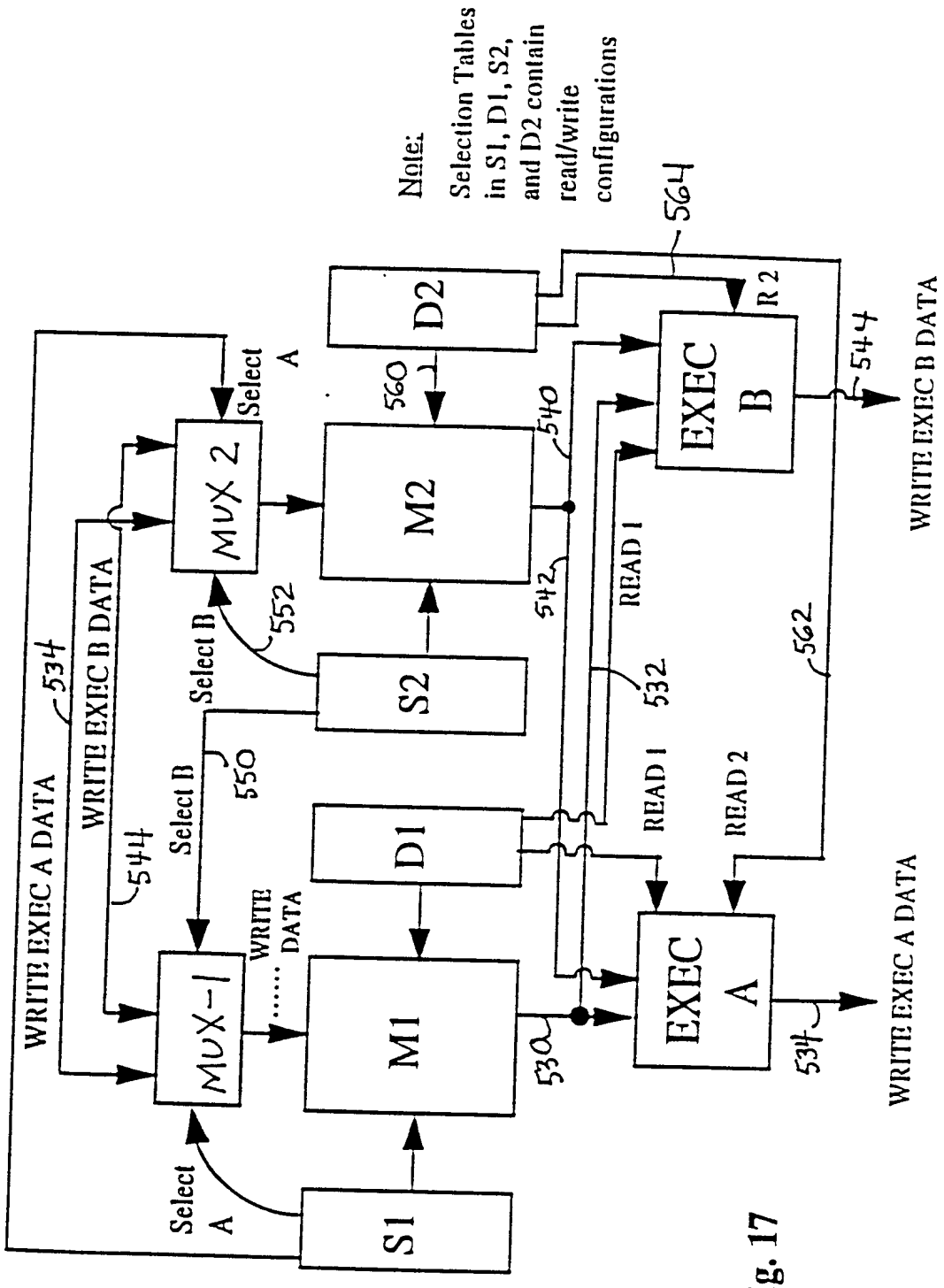

Note:
Selection Tables in S1, D1, S2, and D2 contain read/write configurations
564
D2
560
1
M2
540
542
S2
552
Select B
550
Select A
MUX 2
EXEC B
R 2
544
WRITE EXEC B DATA
READ 1
532
562
READ 2
WRITE EXEC B DATA
534
WRITE EXEC A DATA
544
Select B
D1
WRITE DATA
MUX-1
M1
530
EXEC A
READ 1
READ 2
WRITE EXEC A DATA
534
Select A
S1

Fig. 17

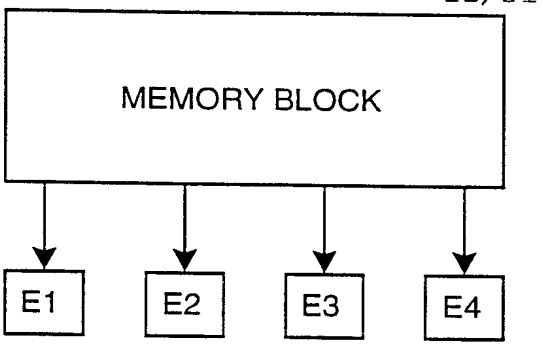


FIG. 18A

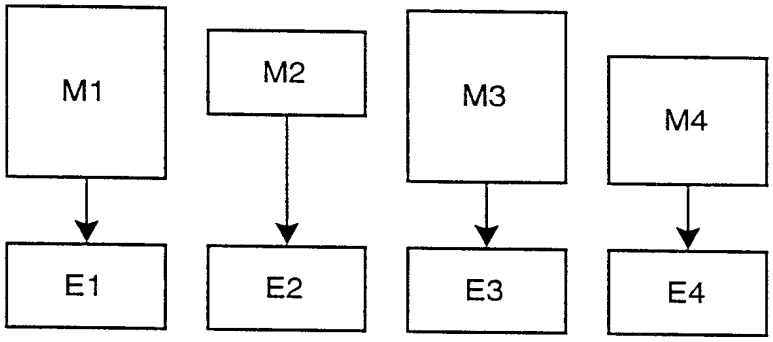


FIG. 18B

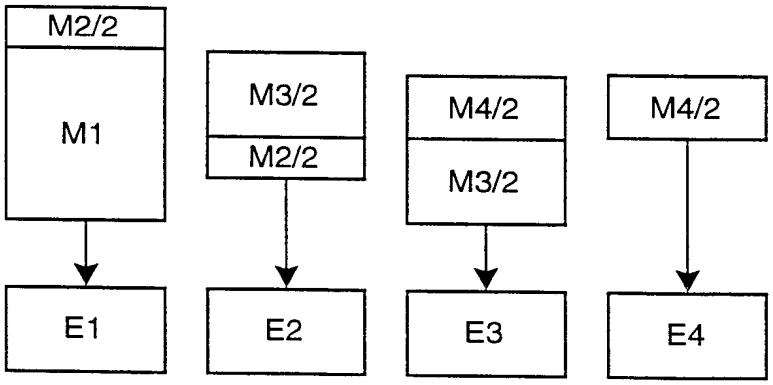


FIG. 18C

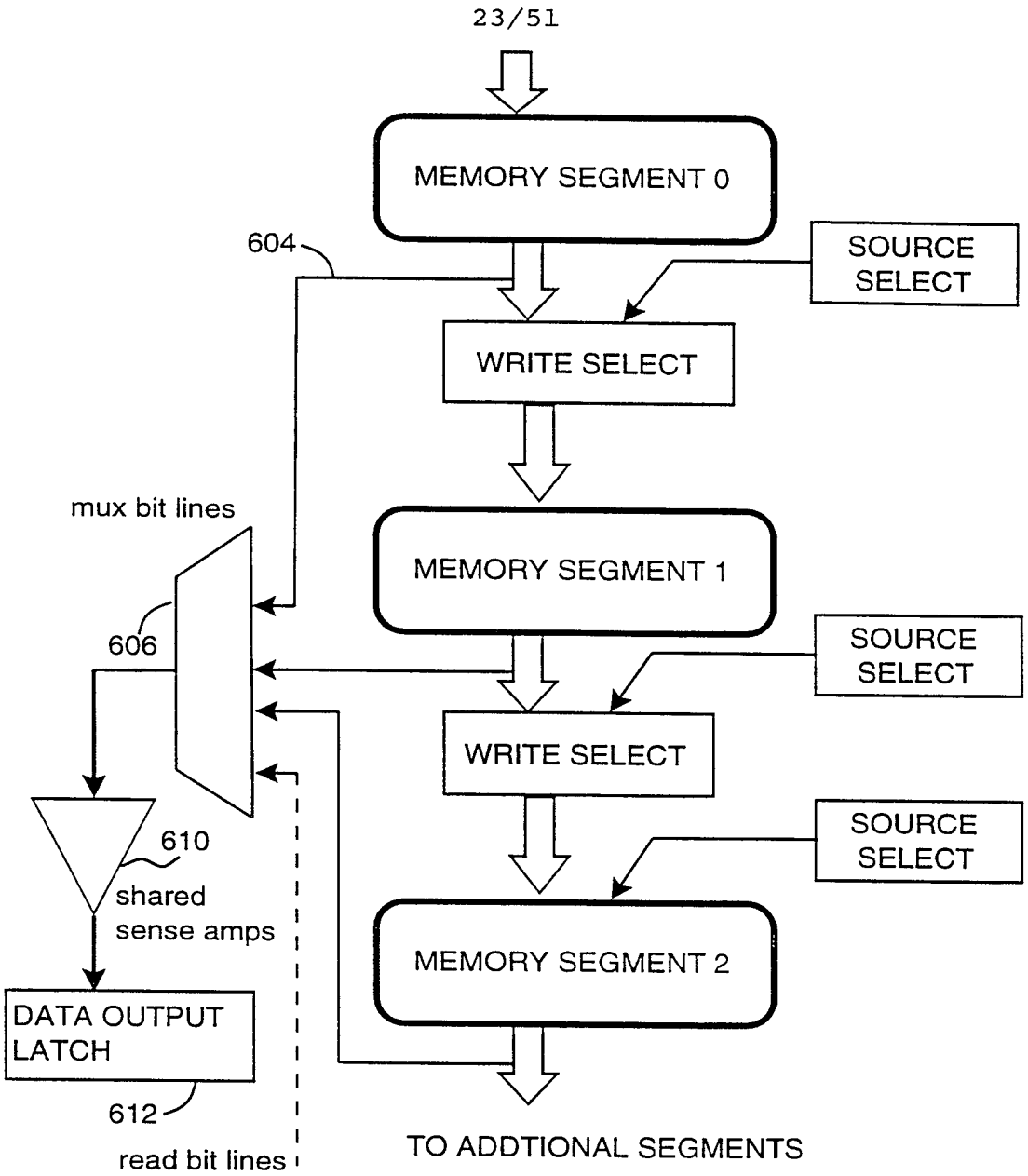

MEMORY SEGMENT 0
604
SOURCE SELECT
WRITE SELECT
mux bit lines
MEMORY SEGMENT 1
606
SOURCE SELECT
WRITE SELECT
610
SOURCE SELECT
shared sense amps
MEMORY SEGMENT 2
DATA OUTPUT LATCH
612
read bit lines
TO ADDTIONAL SEGMENTS

FIG. 19

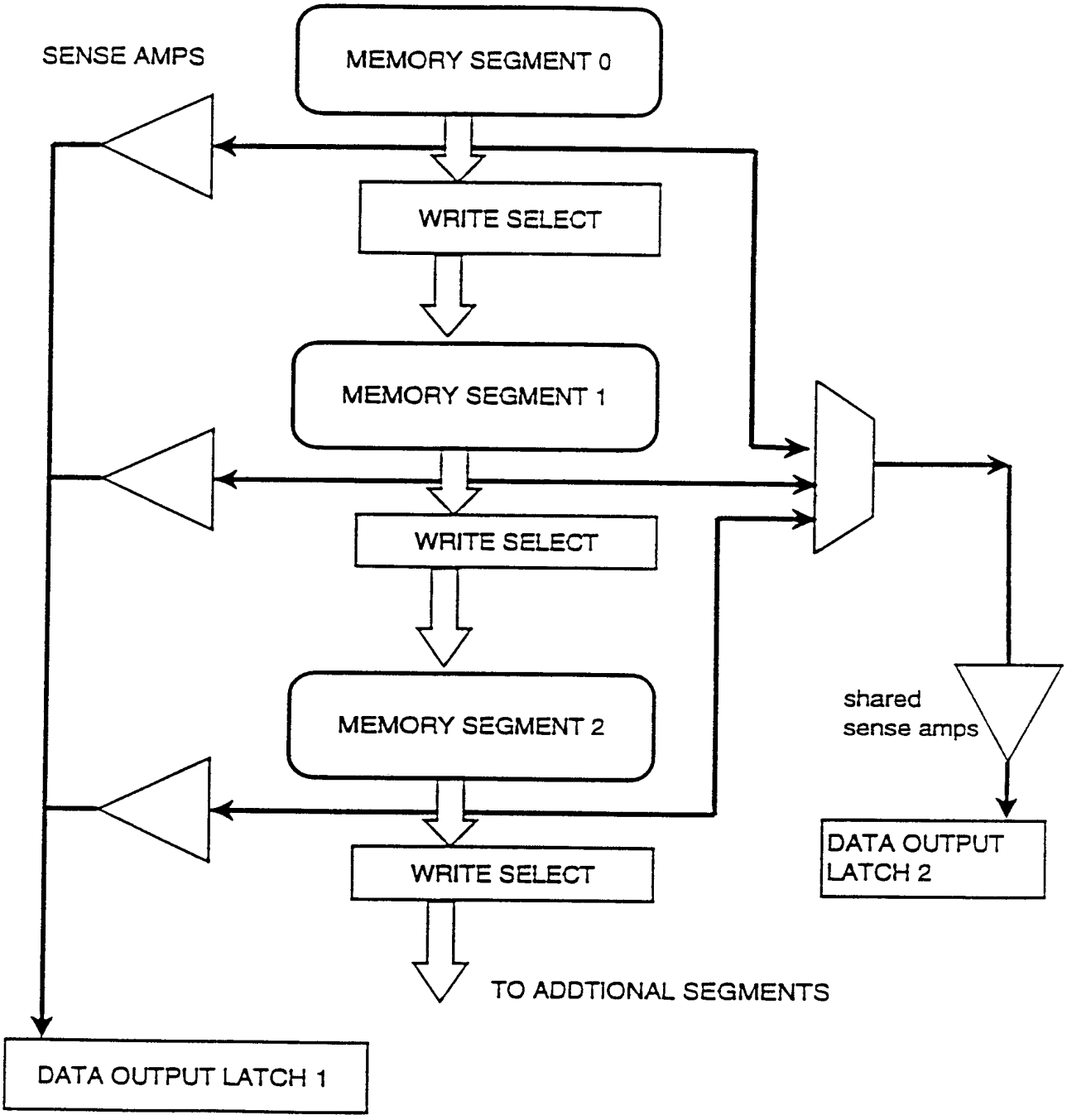

SENSE AMPS
MEMORY SEGMENT 0
WRITE SELECT
MEMORY SEGMENT 1
WRITE SELECT
MEMORY SEGMENT 2
WRITE SELECT
shared
sense amps
DATA OUTPUT
LATCH 2
TO ADDTIONAL SEGMENTS
DATA OUTPUT LATCH 1

FIG. 20

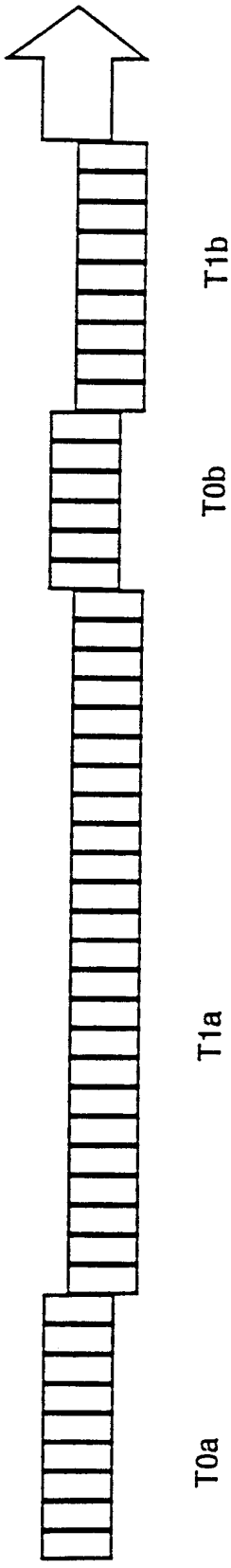


FIG. 21

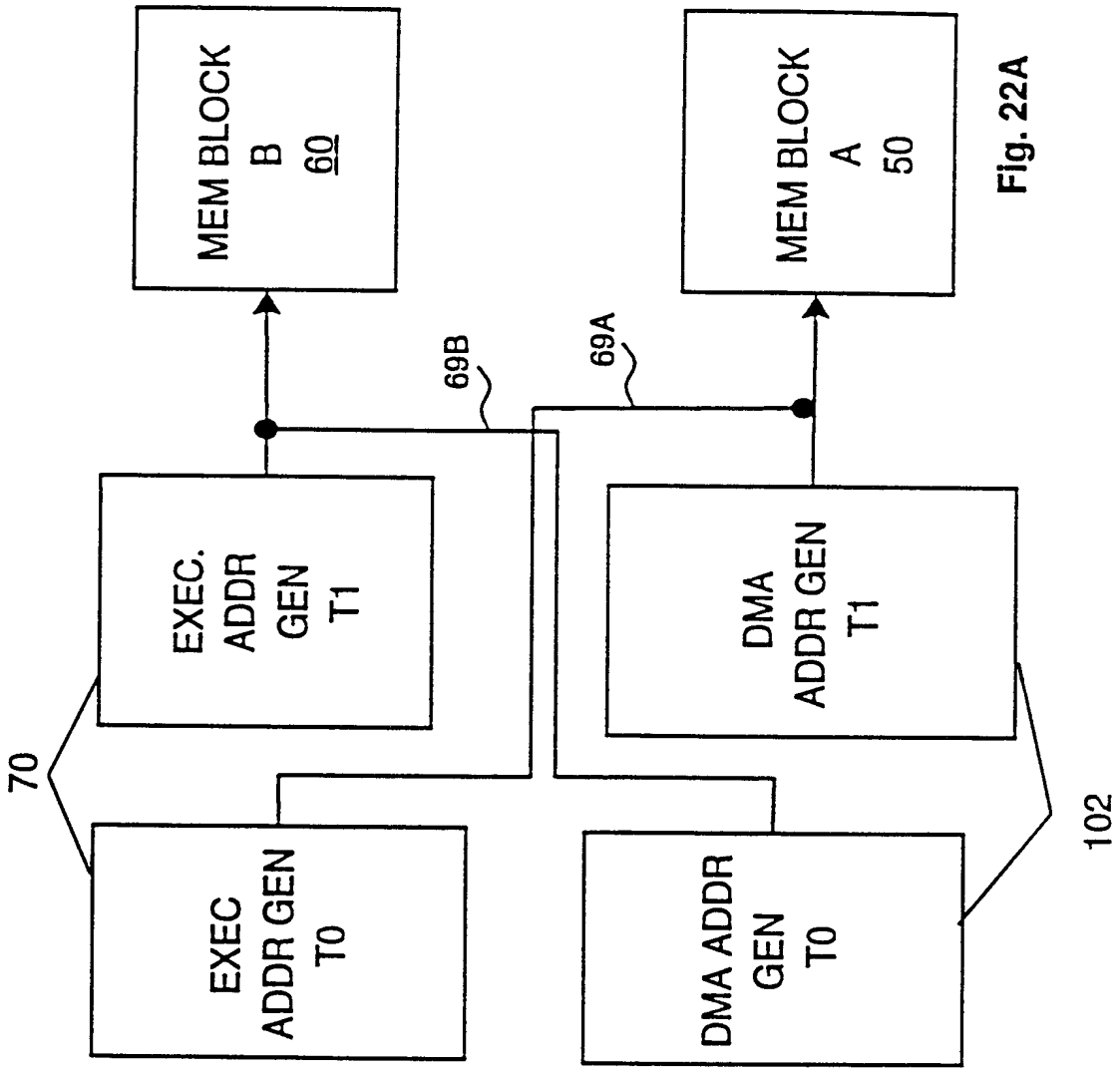

MEM BLOCK B 60
MEM BLOCK A 50
69B
69A
EXEC. ADDR GEN T1
DMA ADDR GEN T1
70
EXEC ADDR GEN T0
DMA ADDR GEN T0
102

Fig. 22A

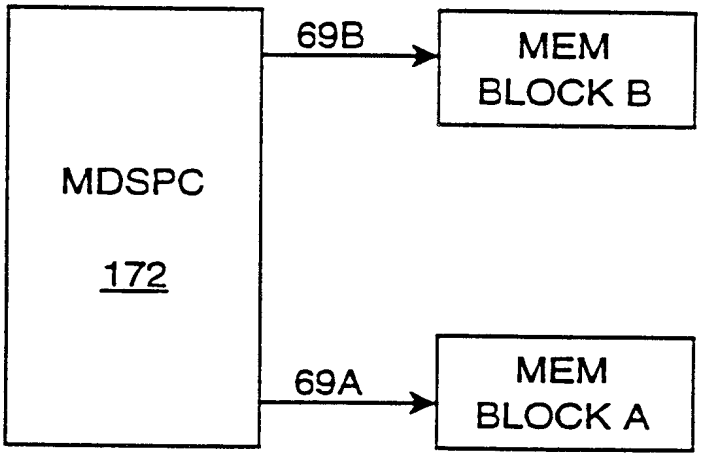


FIG. 22B

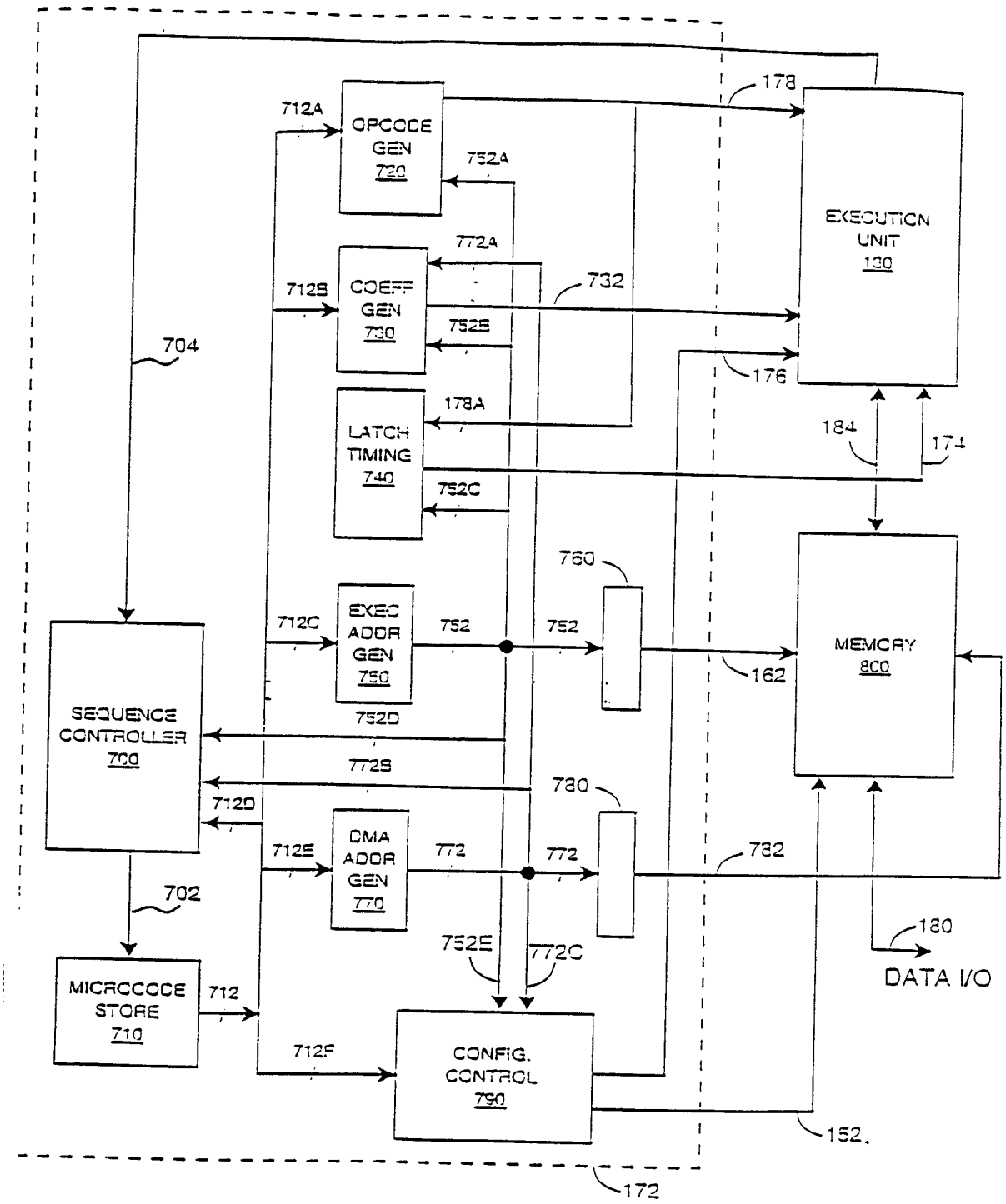

178
712A
OPCODE GEN 720
752A
EXECUTION UNIT 130
772A
712B
COEFF GEN 730
732
752B
704
176
178A
LATCH TIMING 740
184
174
752C
760
712C
EXEC ADDR GEN 750
752
752
MEMORY 800
162
SEQUENCE CONTROLLER 700
752D
772B
780
712D
712E
DMA ADDR GEN 770
772
772
782
702
752E
772C
180
DATA I/O
MICROCODE STORE 710
712
712F
CONFIG. CONTROL 790
152
172

FIG. 23A

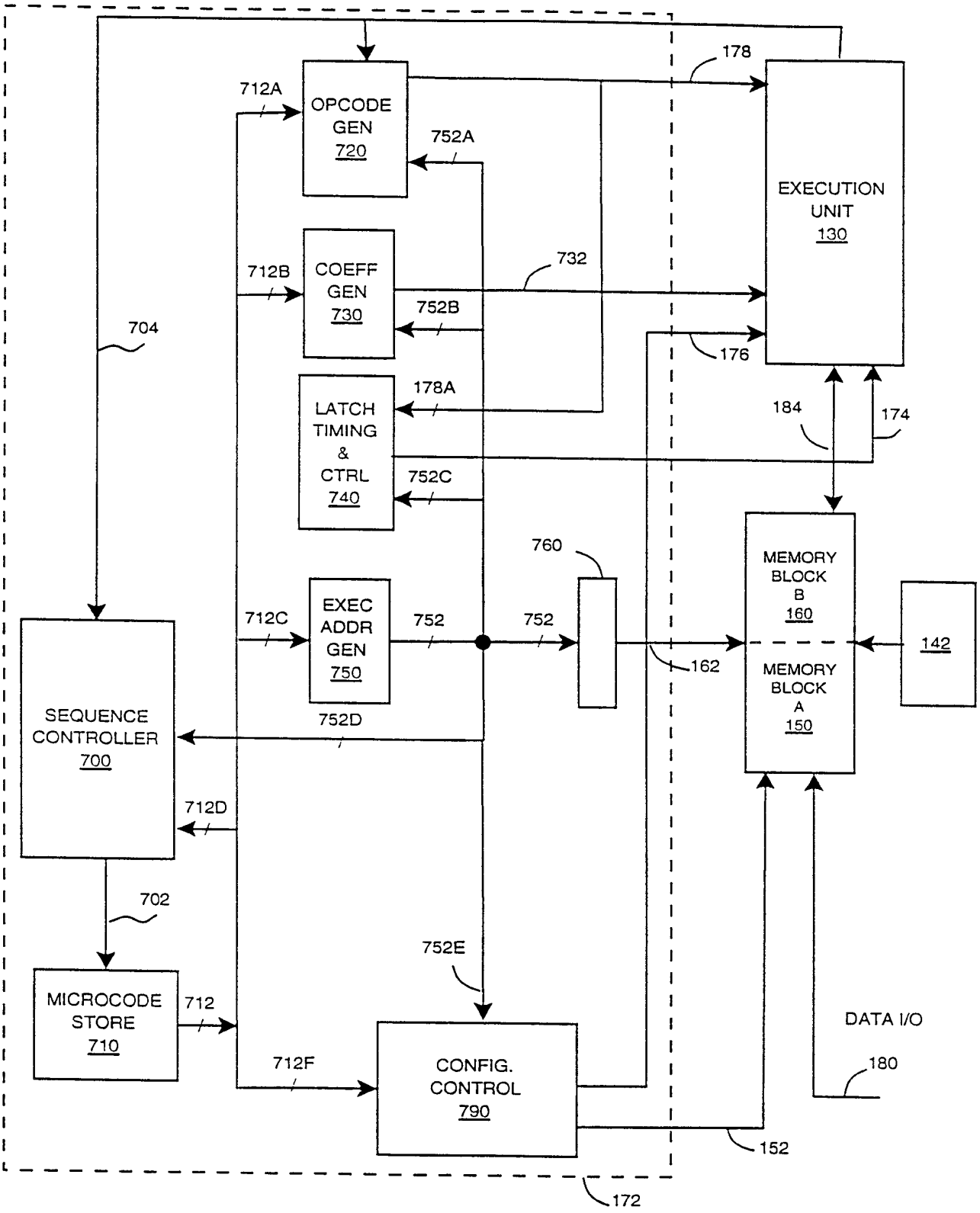

178
712A
OPCODE GEN 720
752A
EXECUTION UNIT 130
712B
COEFF GEN 730
732
752B
704
176
178A
LATCH TIMING & CTRL 740
184
174
752C
760
MEMORY BLOCK B 160
712C
EXEC ADDR GEN 750
752
752
142
162
MEMORY BLOCK A 150
SEQUENCE CONTROLLER 700
752D
712D
702
752E
MICROCODE STORE 710
712
DATA I/O
712F
CONFIG. CONTROL 790
180
152
172

FIG. 23B

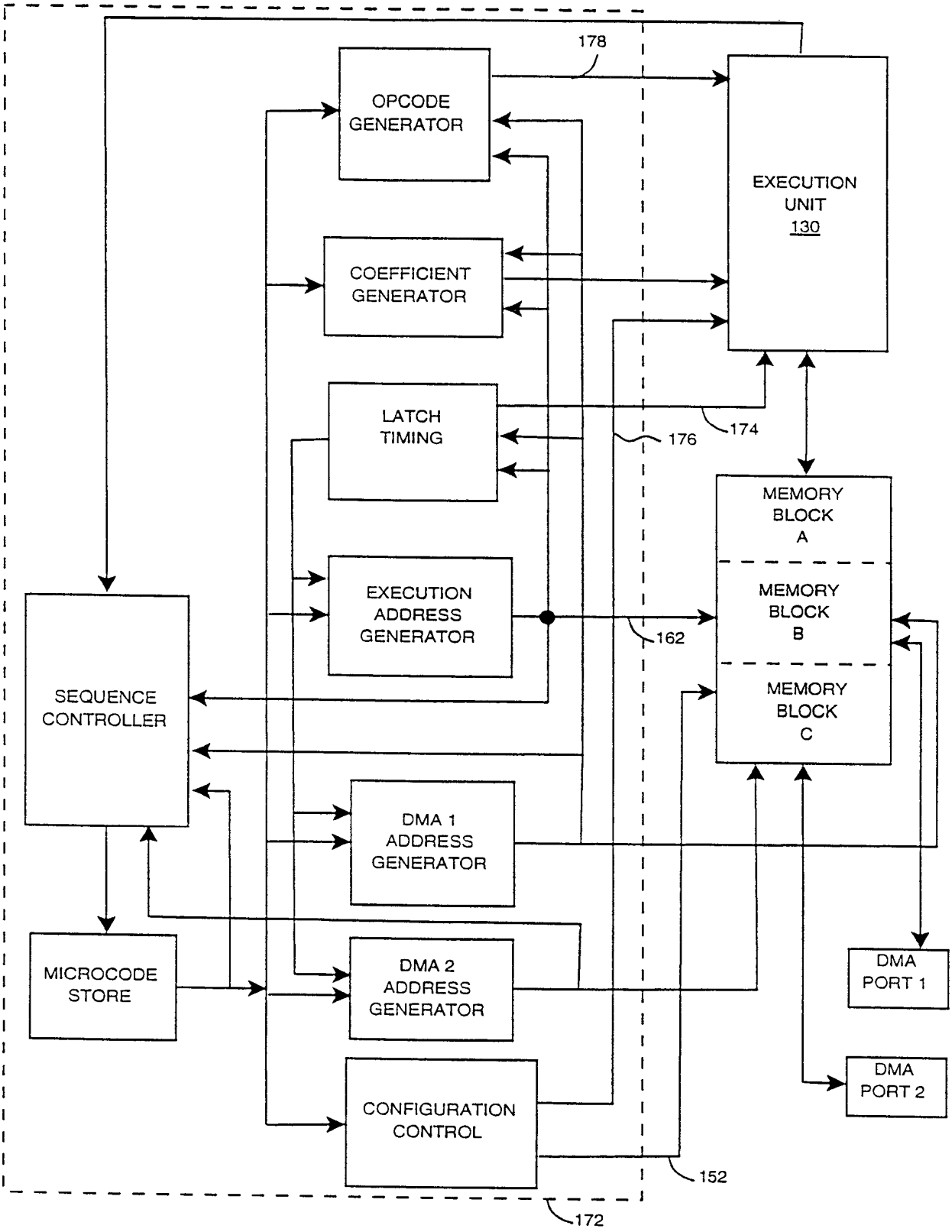

178
OPCODE GENERATOR
EXECUTION UNIT 130
COEFFICIENT GENERATOR
LATCH TIMING
176
174
MEMORY BLOCK A
MEMORY BLOCK B
MEMORY BLOCK C
EXECUTION ADDRESS GENERATOR
162
SEQUENCE CONTROLLER
DMA 1 ADDRESS GENERATOR
MICROCODE STORE
DMA 2 ADDRESS GENERATOR
DMA PORT 1
DMA PORT 2
CONFIGURATION CONTROL
152
172

FIG. 23C

ADDRESS	FUNCTION
0x710	FFT RADIX 2
0x805	FIR FILTER
0x2100	CONVOLUTION

FIG. 24A

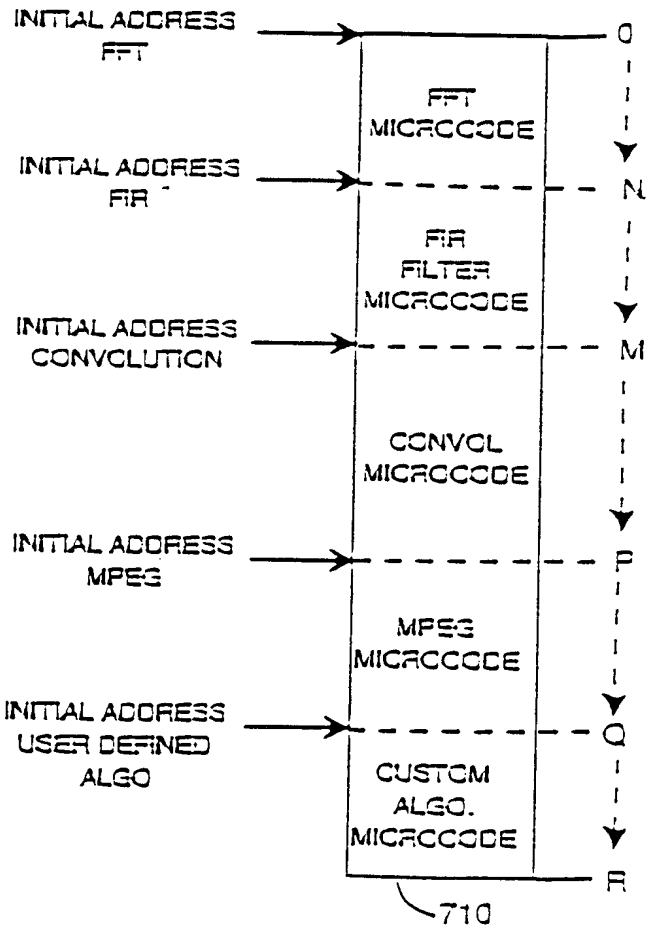


FIG. 24B

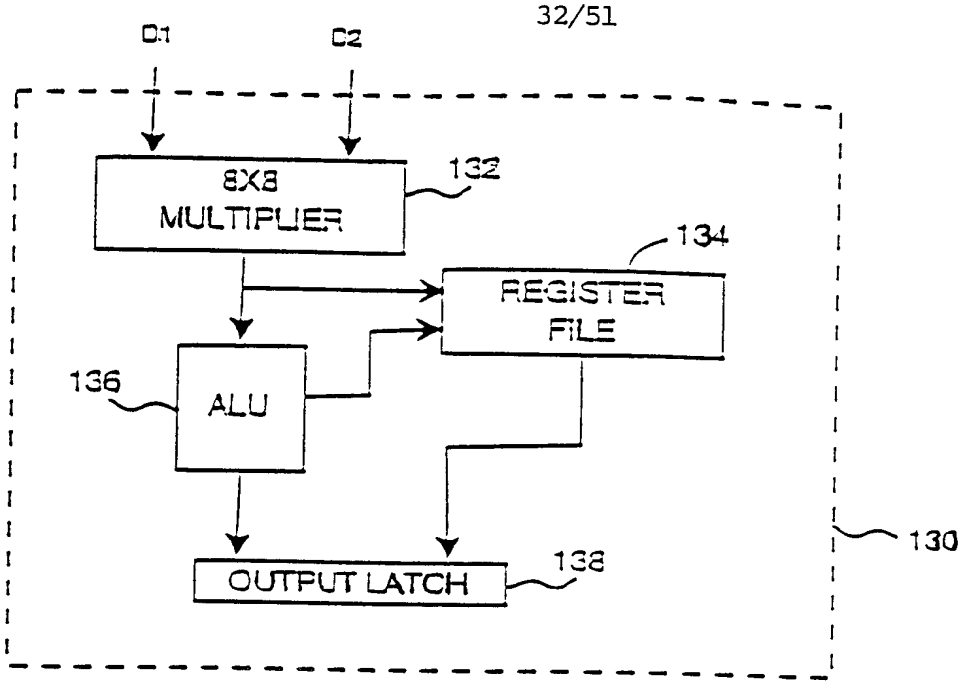

C1
C2
8X8
MULTIPLIER
132
134
REGISTER
FILE
136
ALU
OUTPUT LATCH
138
130

FIG. 25

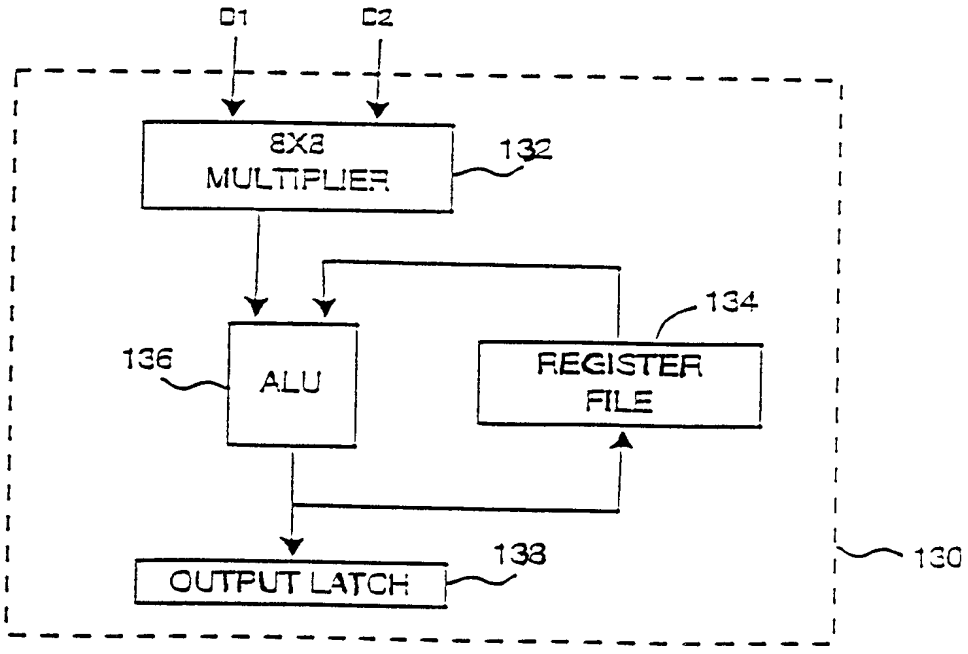

C1
C2
8X8
MULTIPLIER
132
134
136
ALU
REGISTER
FILE
138
OUTPUT LATCH
130

FIG. 26

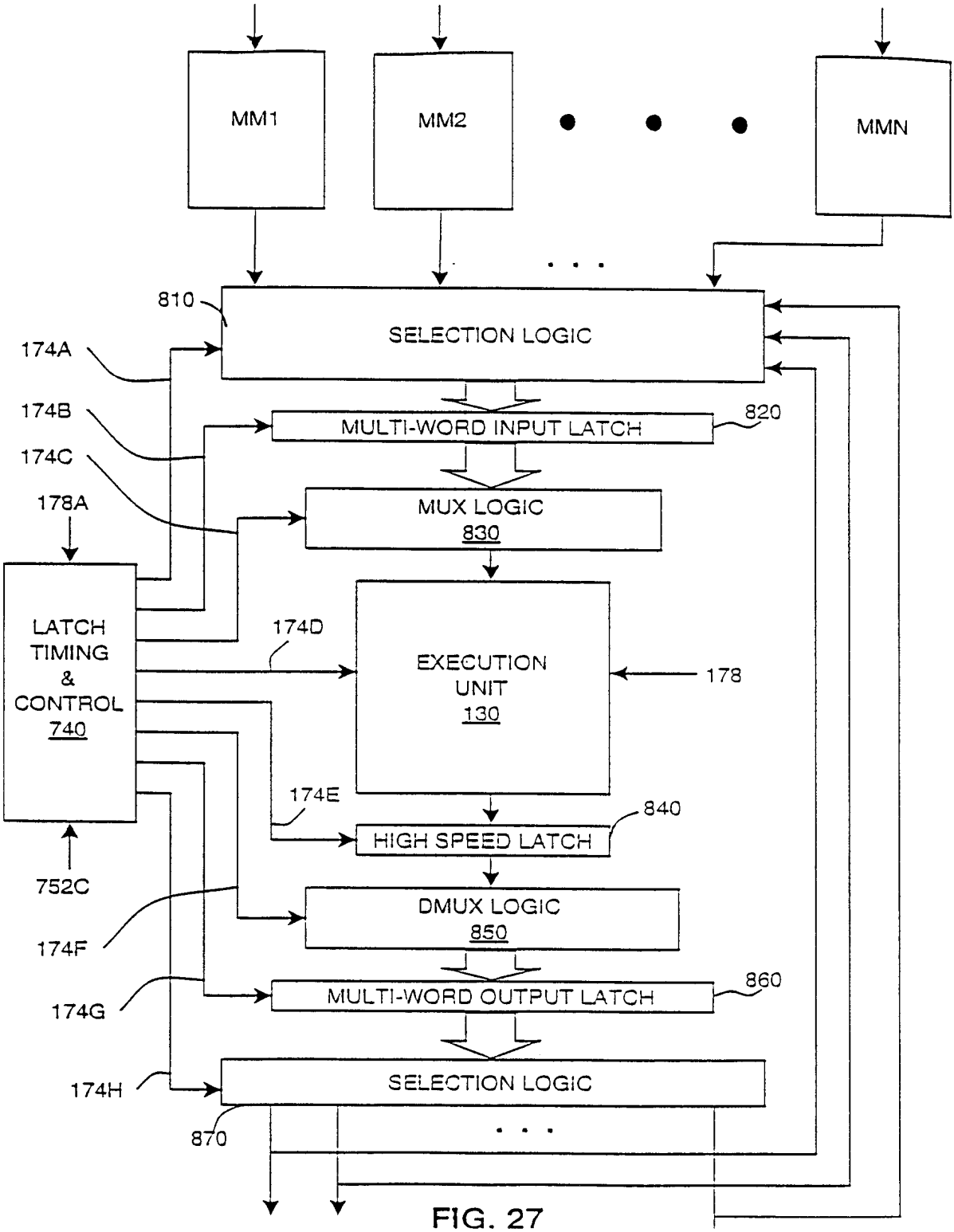

MM1
MM2
MMN
810
SELECTION LOGIC
174A
174B
MULTI-WORD INPUT LATCH
820
174C
178A
MUX LOGIC
830
LATCH
TIMING
&
CONTROL
740
174D
EXECUTION
UNIT
130
178
174E
840
HIGH SPEED LATCH
752C
DMUX LOGIC
850
174F
174G
MULTI-WORD OUTPUT LATCH
860
174H
SELECTION LOGIC
870

FIG. 27

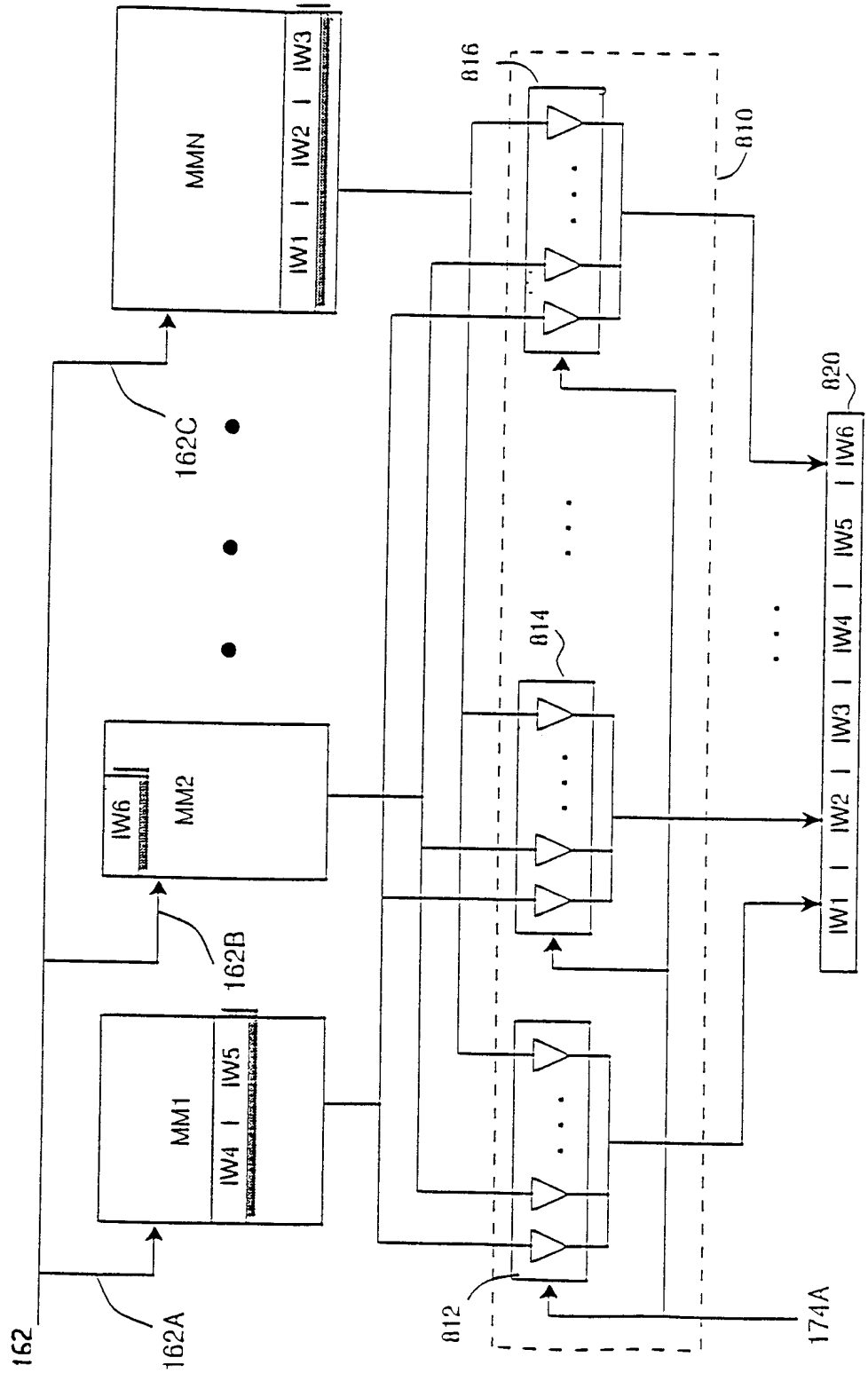

MMN
IW1
IW2
IW3
MM2
IW6
MM1
IW4
IW5
162
162A
162B
162C
812
814
816
810
820
174A
IW1
IW2
IW3
IW4
IW5
IW6

FIG. 28

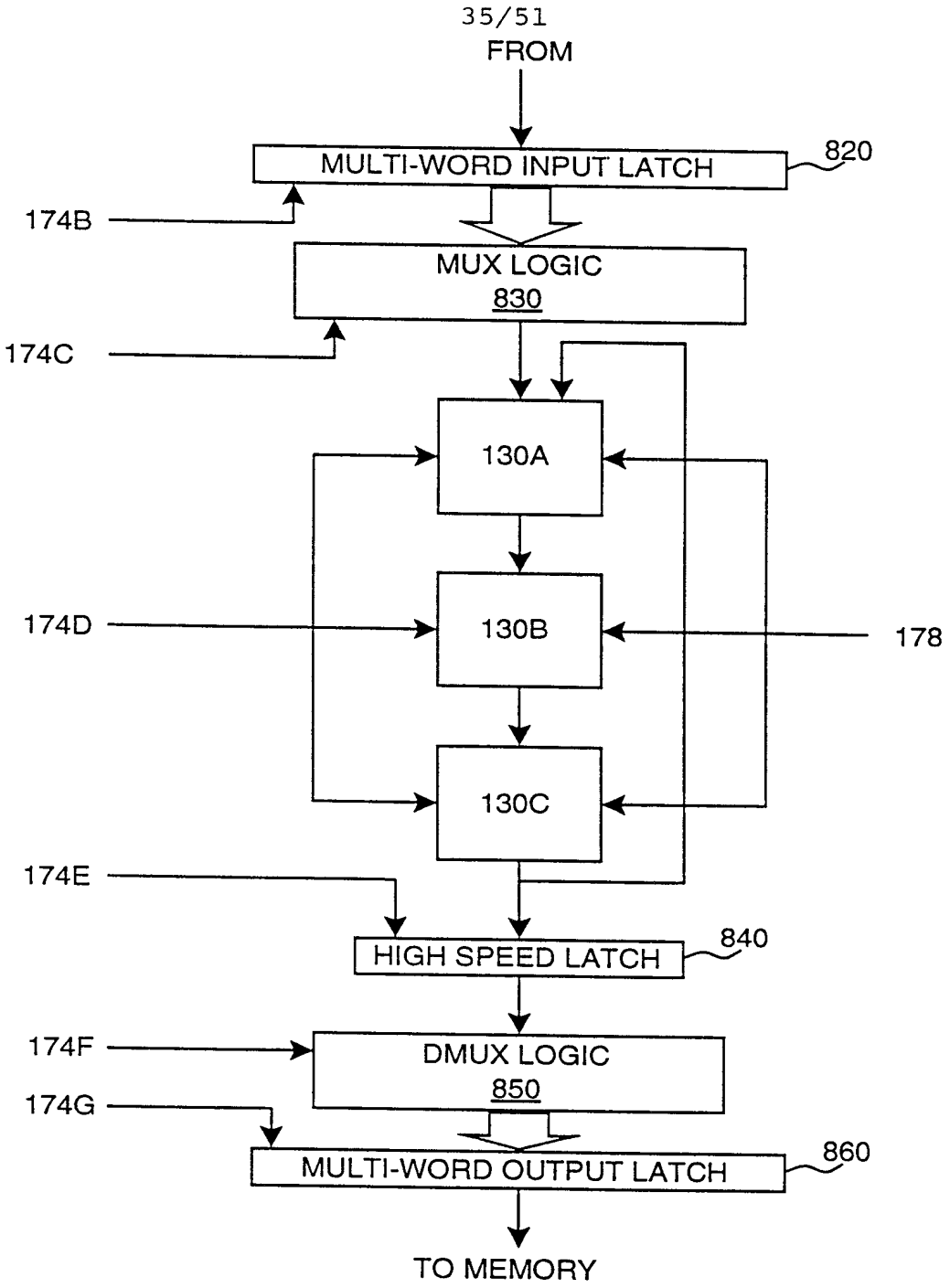

35/51
FROM
MULTI-WORD INPUT LATCH
820
174B
MUX LOGIC
830
174C
130A
130B
174D
178
130C
174E
HIGH SPEED LATCH
840
174F
DMUX LOGIC
850
174G
MULTI-WORD OUTPUT LATCH
860
TO MEMORY

FIG. 29

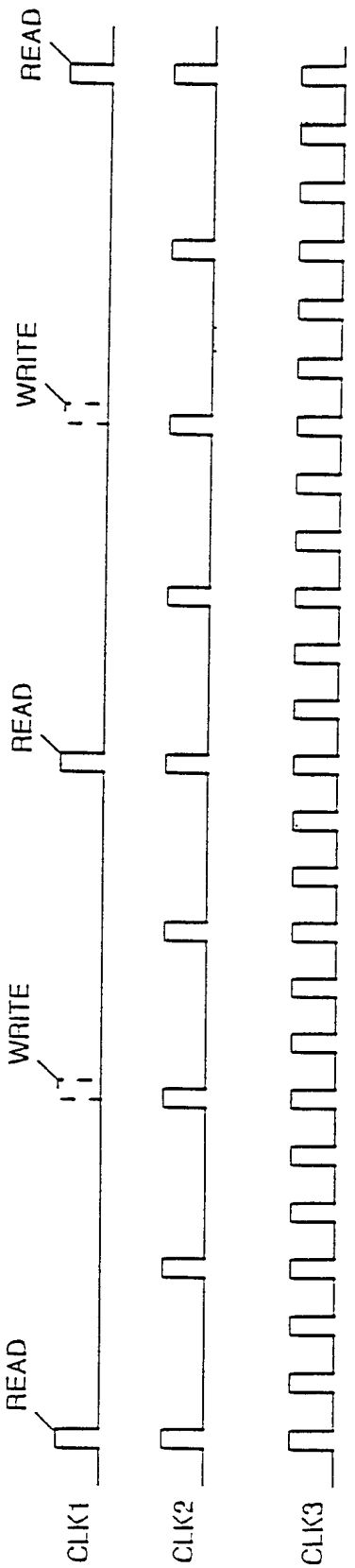


FIG. 30A

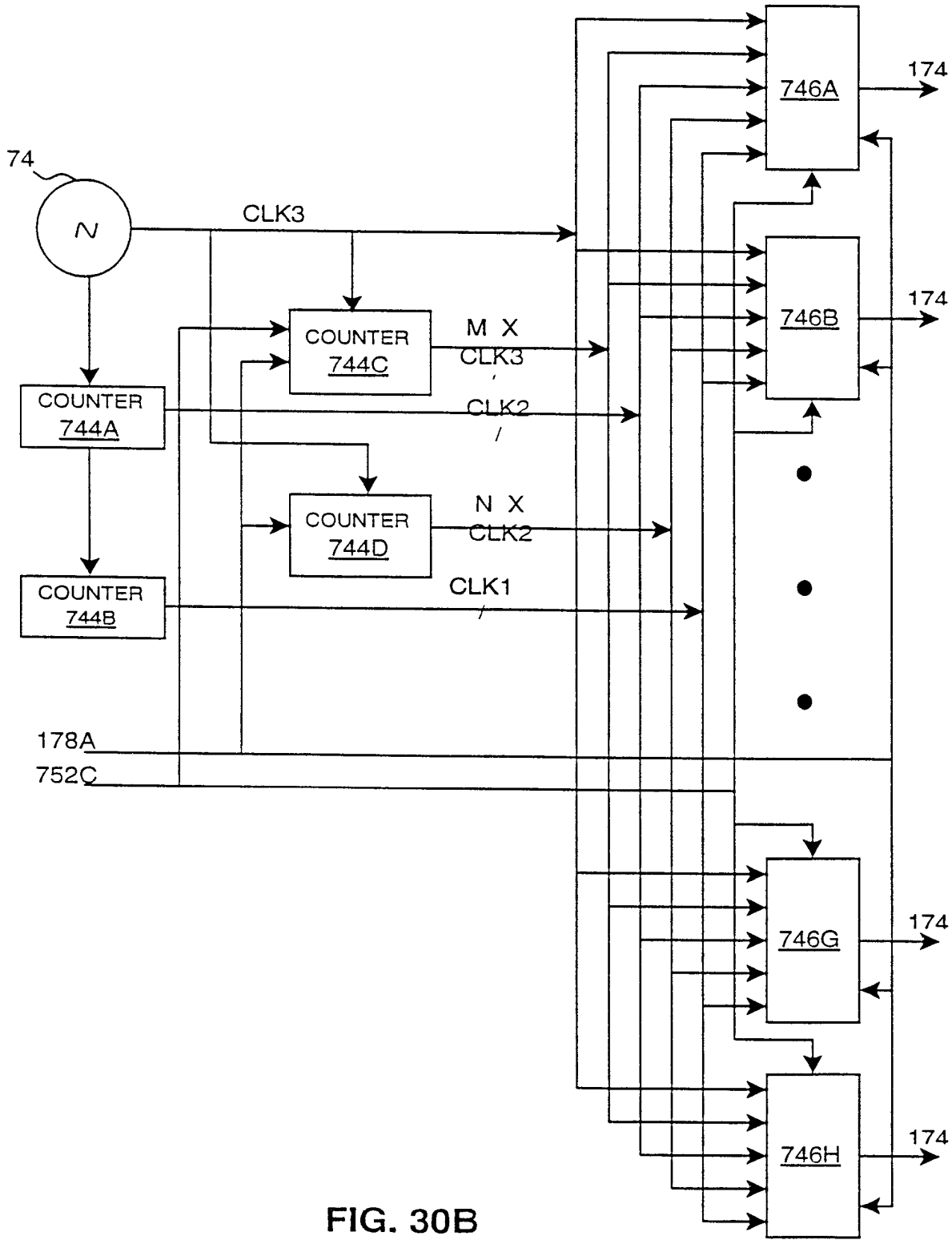

74
N
CLK3
COUNTER 744A
COUNTER 744B
COUNTER 744C
COUNTER 744D
M X CLK3
CLK2
N X CLK2
CLK1
178A
752C
746A
746B
746G
746H
174
174
174
174

FIG. 30B

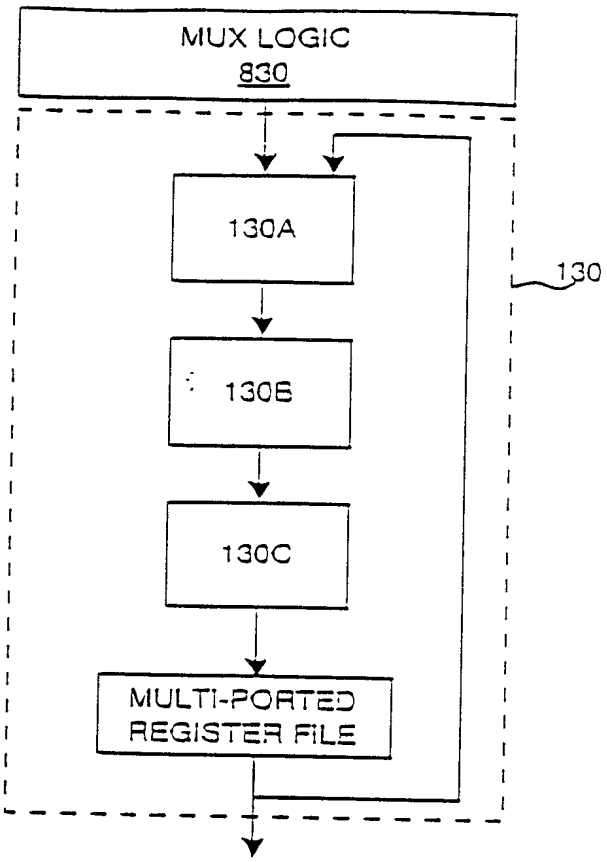

MUX LOGIC
830
130A
130B
130C
MULTI-PORTED
REGISTER FILE
130

FIG. 31

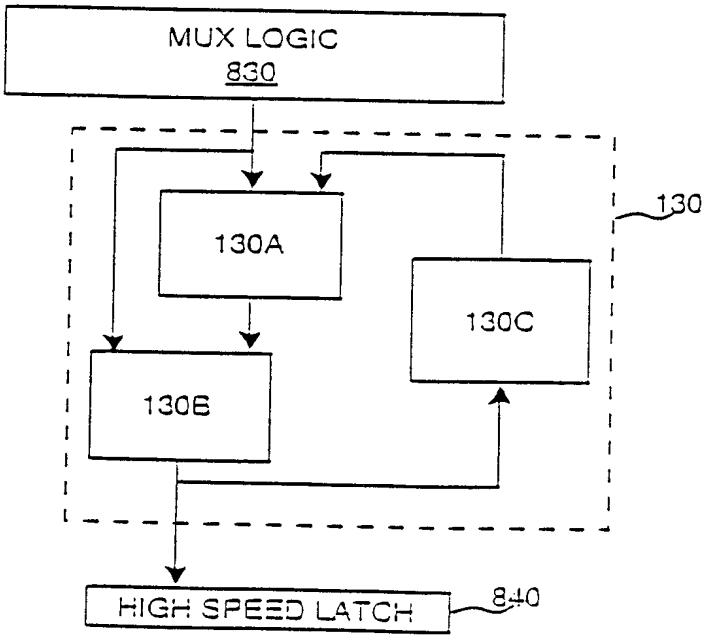

MUX LOGIC
830
130A
130B
130C
130
HIGH SPEED LATCH
840

FIG. 32

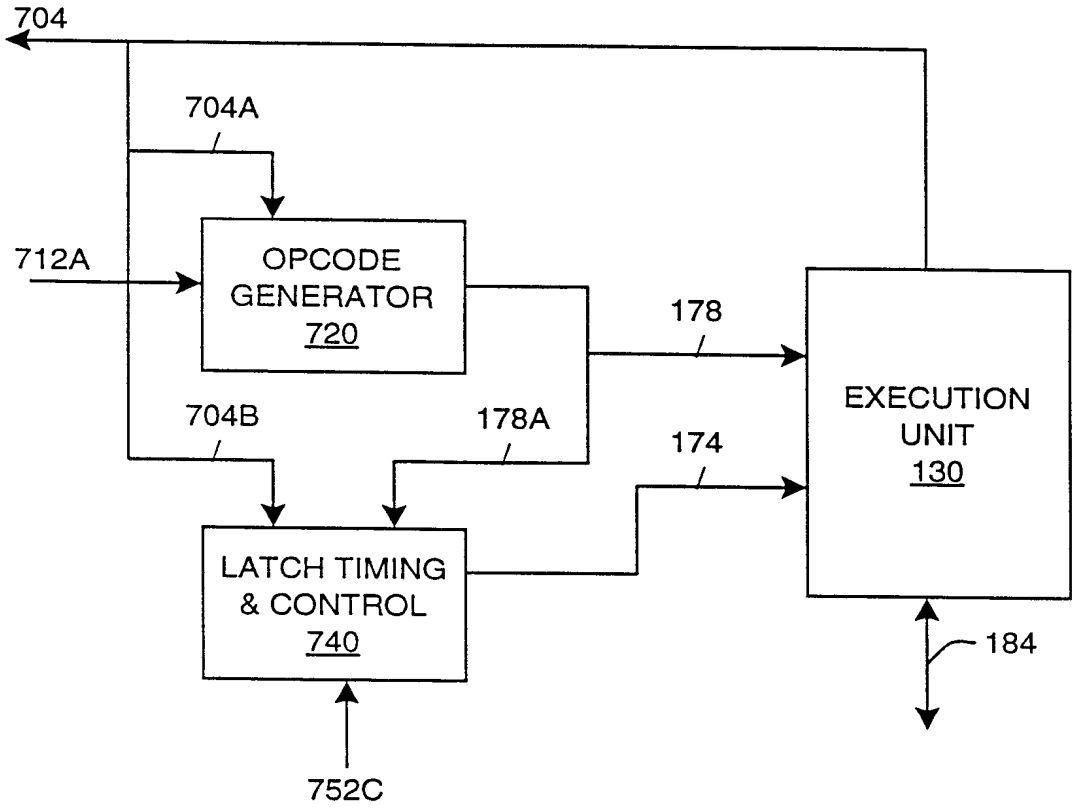

704
704A
712A
OPCODE
GENERATOR
720
178
EXECUTION
UNIT
130
704B
178A
174
LATCH TIMING
& CONTROL
740
184
752C

FIG. 33

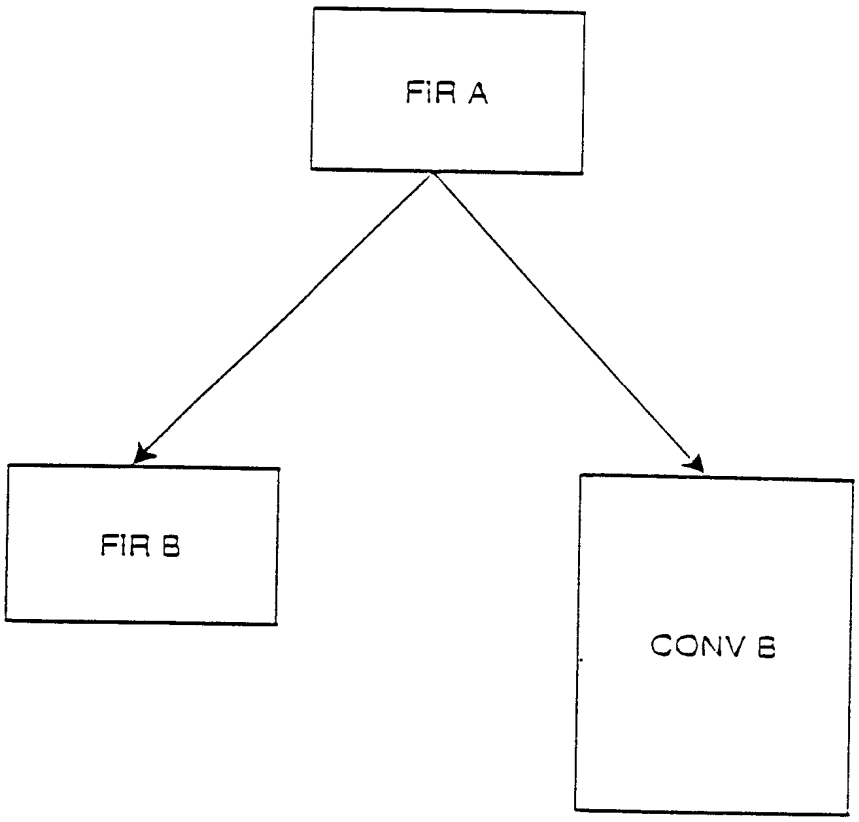


FIG. 34

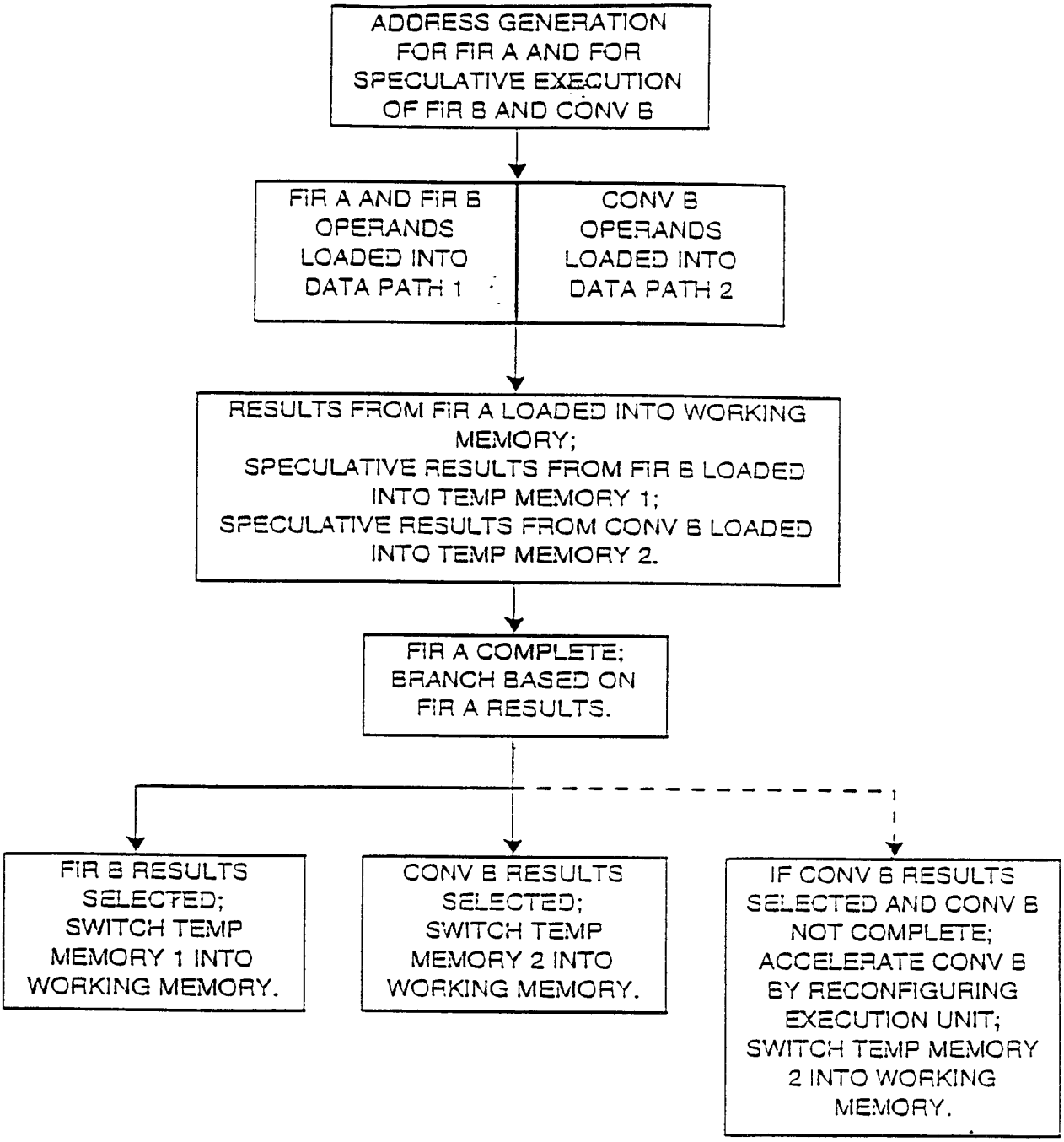

ADDRESS GENERATION FOR FIR A AND FOR SPECULATIVE EXECUTION OF FIR B AND CONV B
FIR A AND FIR B OPERANDS LOADED INTO DATA PATH 1
CONV B OPERANDS LOADED INTO DATA PATH 2
RESULTS FROM FIR A LOADED INTO WORKING MEMORY; SPECULATIVE RESULTS FROM FIR B LOADED INTO TEMP MEMORY 1; SPECULATIVE RESULTS FROM CONV B LOADED INTO TEMP MEMORY 2.
FIR A COMPLETE; BRANCH BASED ON FIR A RESULTS.
FIR B RESULTS SELECTED; SWITCH TEMP MEMORY 1 INTO WORKING MEMORY.
CONV B RESULTS SELECTED; SWITCH TEMP MEMORY 2 INTO WORKING MEMORY.
IF CONV B RESULTS SELECTED AND CONV B NOT COMPLETE; ACCELERATE CONV B BY RECONFIGURING EXECUTION UNIT; SWITCH TEMP MEMORY 2 INTO WORKING MEMORY.

FIG. 35

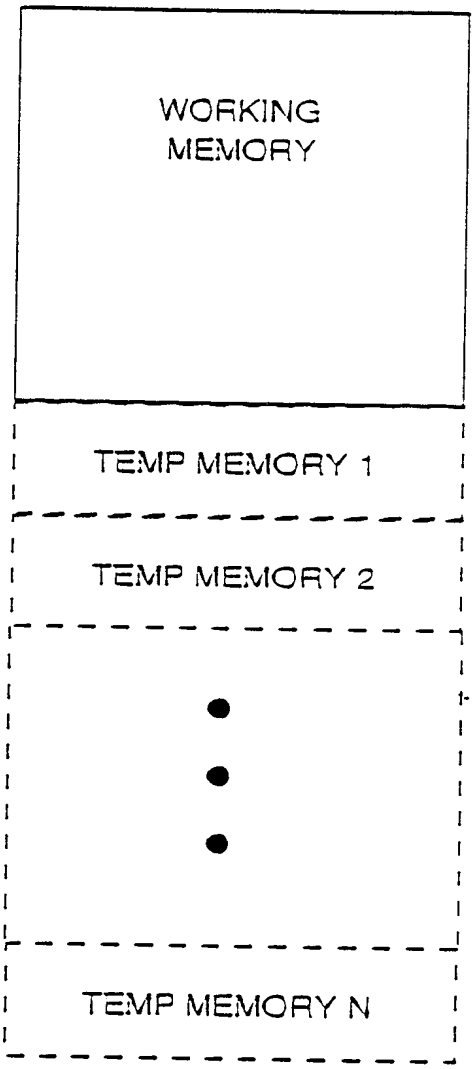


FIG. 36

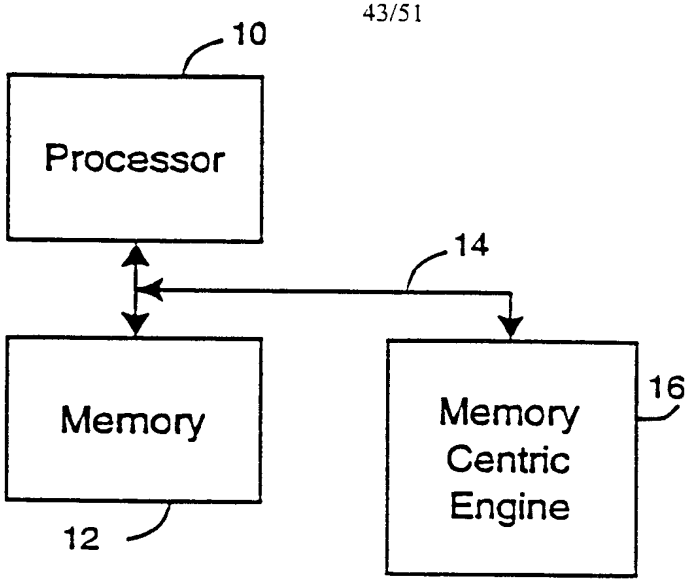


FIG. 37

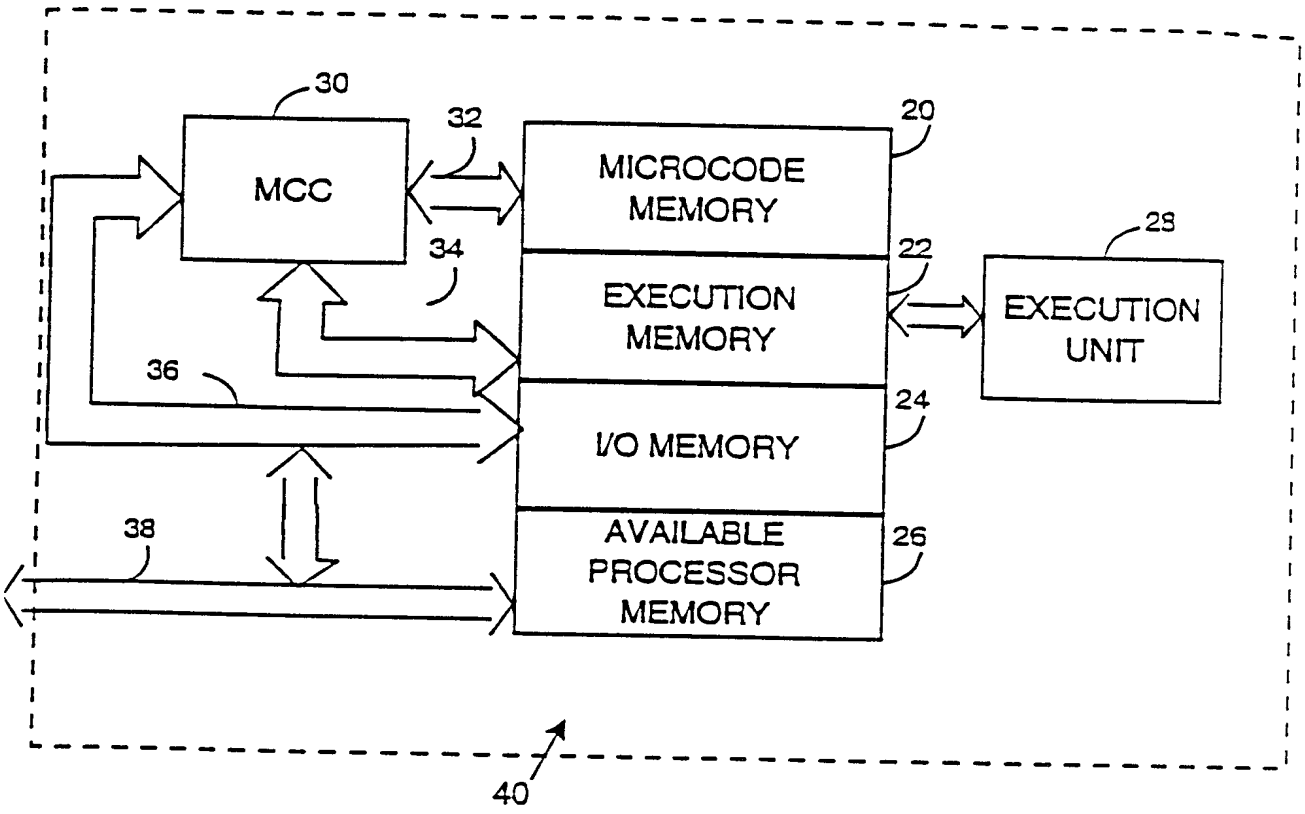

30
MCC
32
MICROCODE MEMORY
20
34
EXECUTION MEMORY
22
28
EXECUTION UNIT
36
I/O MEMORY
24
38
AVAILABLE PROCESSOR MEMORY
26
40

FIG. 38

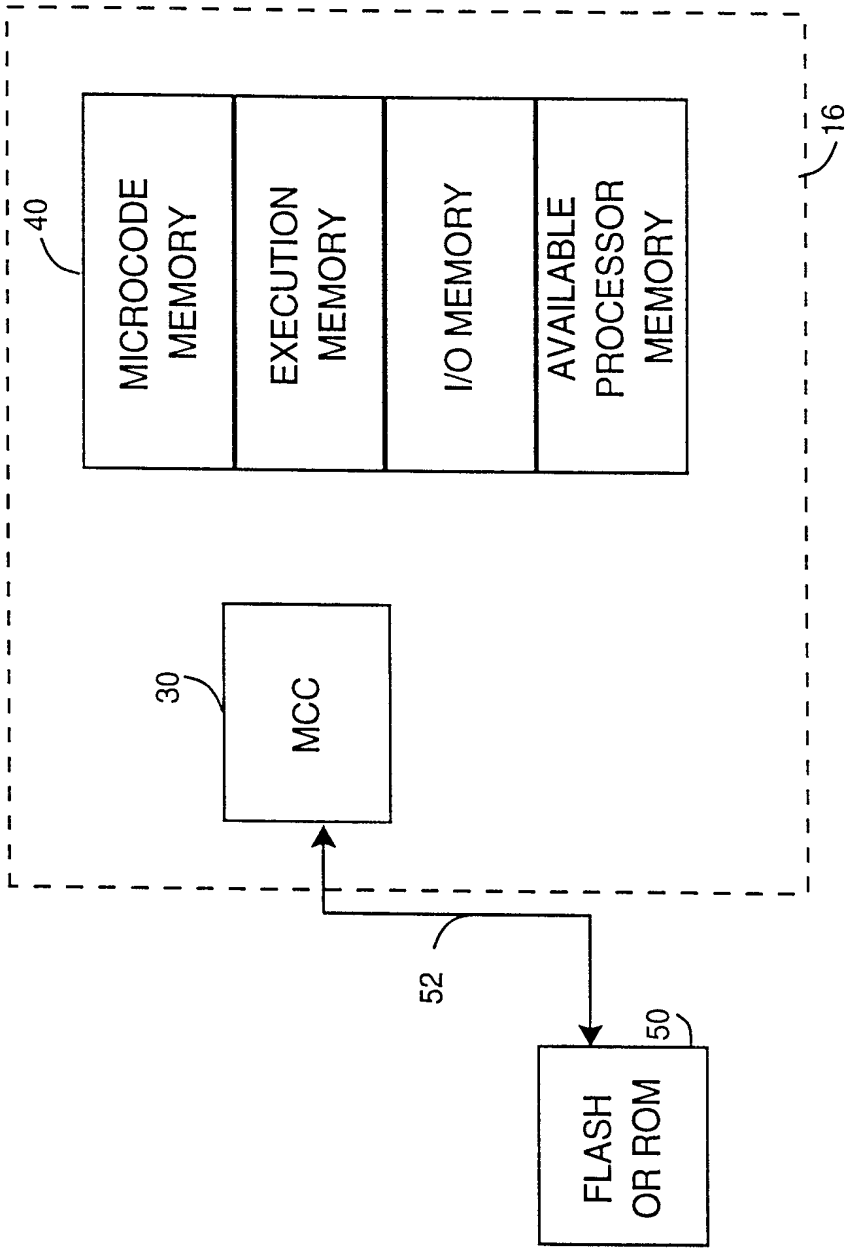

16
40
MICROCODE MEMORY
EXECUTION MEMORY
I/O MEMORY
AVAILABLE PROCESSOR MEMORY
30
MCC
52
50
FLASH OR ROM

FIG. 39

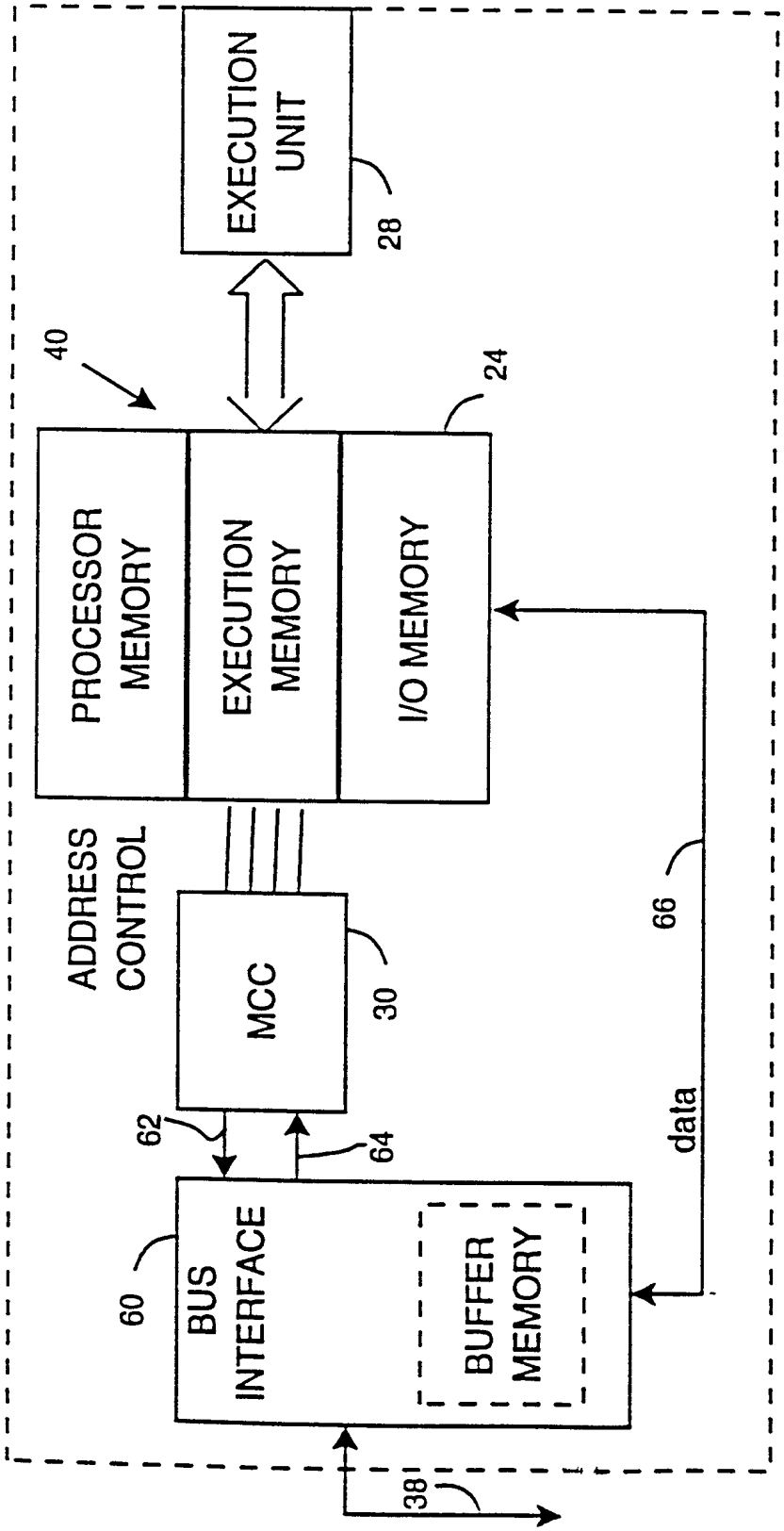
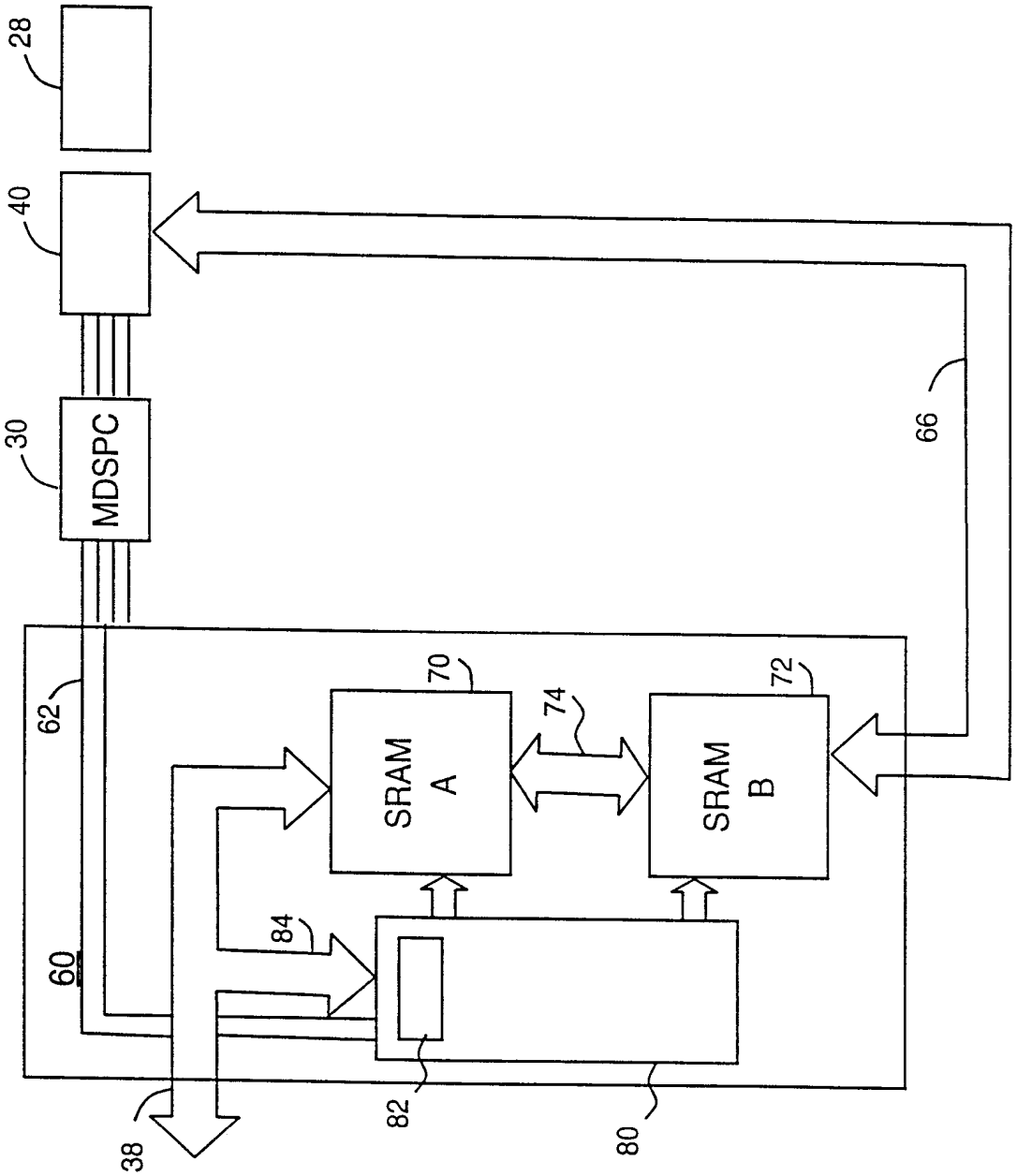

EXECUTION UNIT
28
40
24
PROCESSOR MEMORY
EXECUTION MEMORY
I/O MEMORY
ADDRESS CONTROL
MCC
30
66
62
64
data
60
BUS INTERFACE
BUFFER MEMORY
38

FIG. 40

FIG. 41



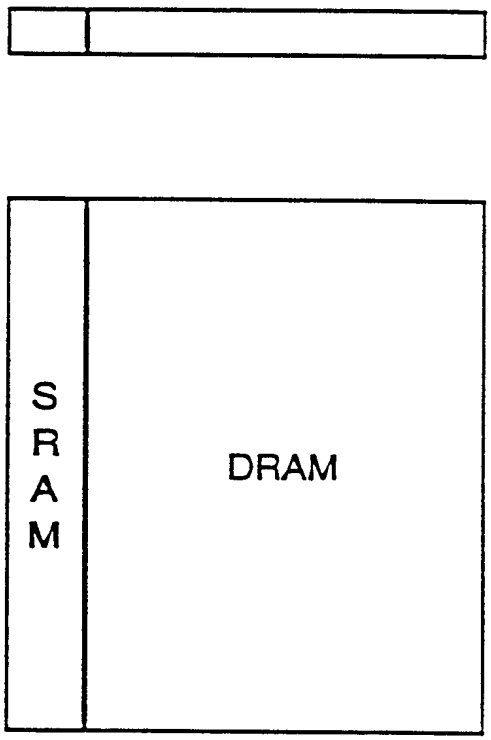


FIG. 42

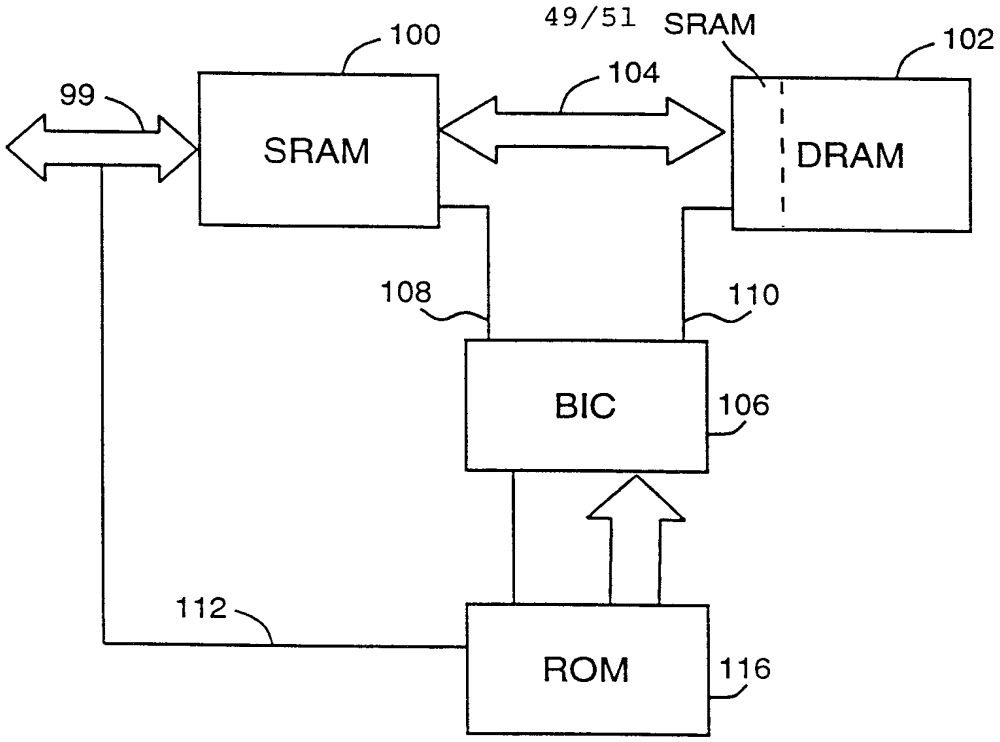


FIG. 43

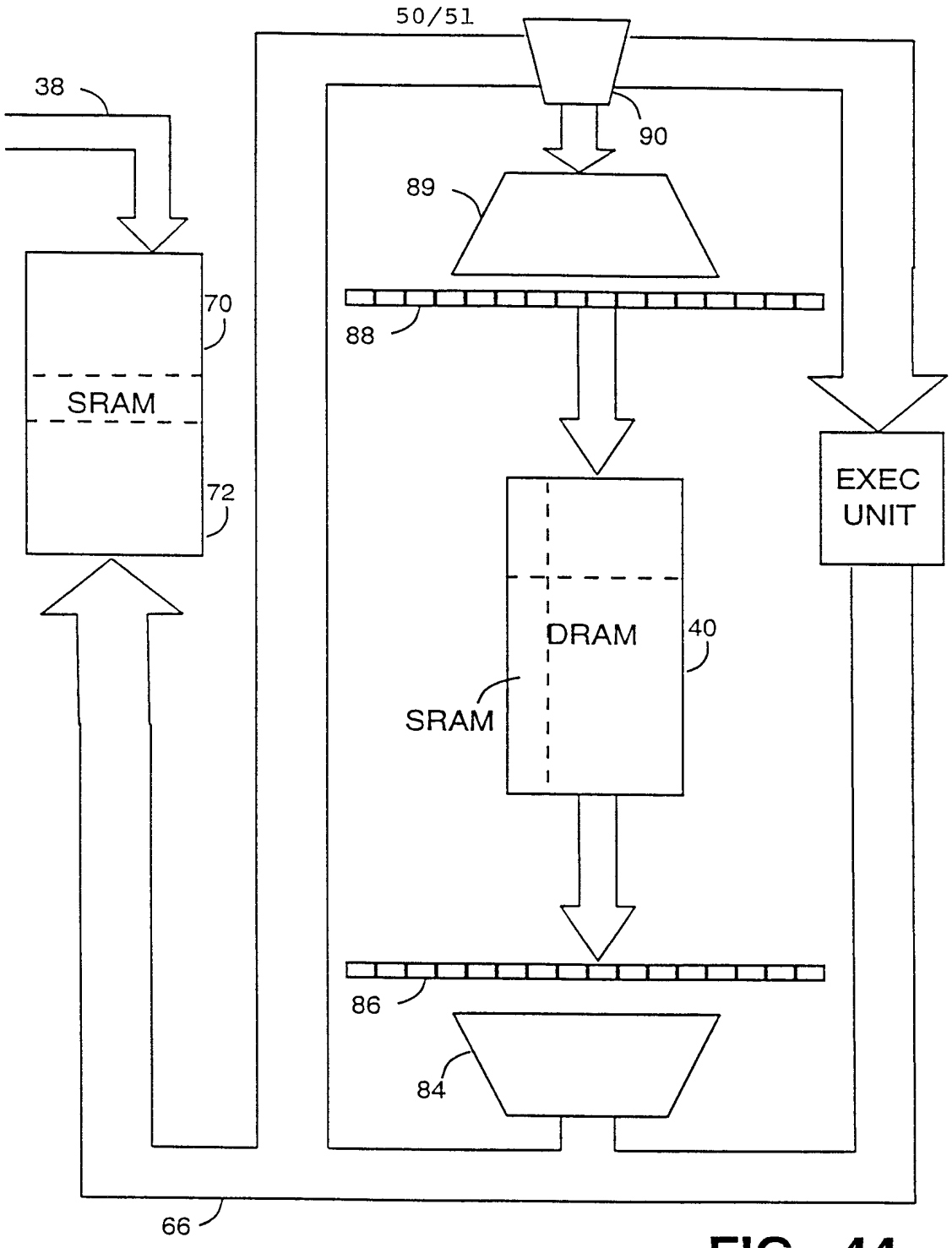


FIG. 44

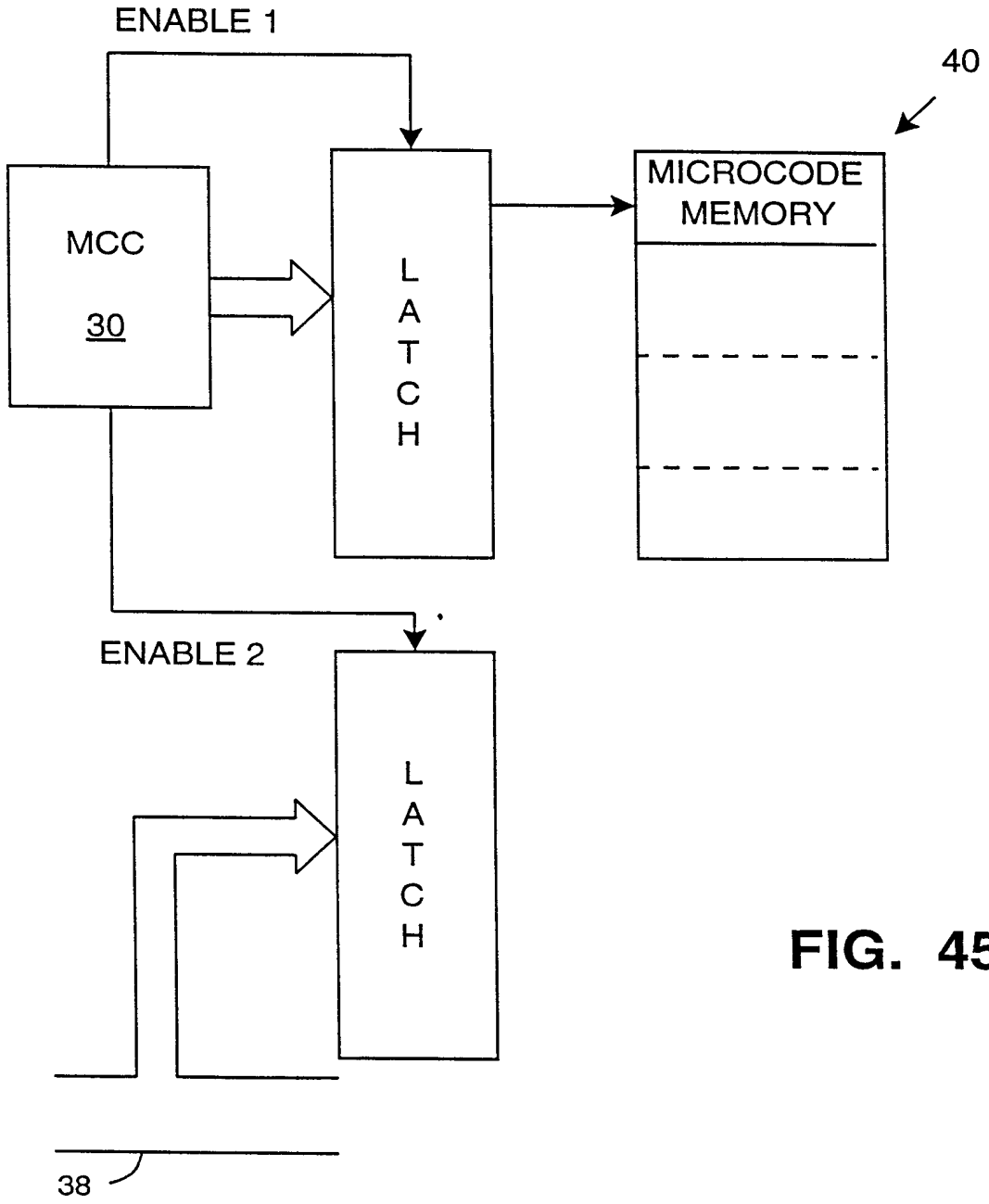


FIG. 45

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 9/38, 9/24, 13/16, G11C 7/00

A3

(11) International Publication Number: **WO 98/55932**

(43) International Publication Date: 10 December 1998 (10.12.98)

(21) International Application Number: PCT/US98/10549

(22) International Filing Date: 22 May 1998 (22.05.98)

(30) Priority Data:

08/869,148	4 June 1997 (04.06.97)	US
08/869,277	4 June 1997 (04.06.97)	US

(71)(72) Applicant and Inventor: RUBINSTEIN, Richard [US/US]; 3418 N.E. 83rd Avenue, Vancouver, WA 98662 (US).

(74) Agents: STOLOWITZ, Micah, D. et al.; 1030 S.W. Morrison, Portland, OR 97205 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

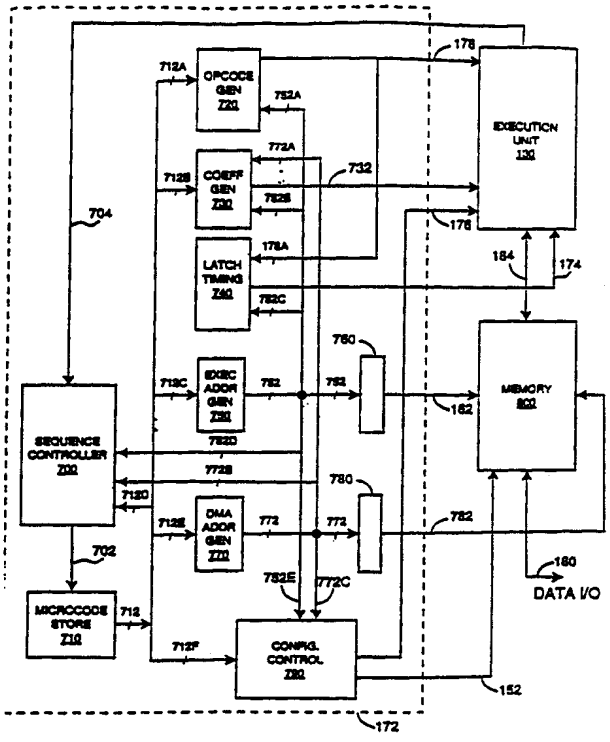
Published
With international search report.

(88) Date of publication of the international search report:
12 August 1999 (12.08.99)

(54) Title: PROCESSOR INTERFACING TO MEMORY MAPPED COMPUTING ENGINE

(57) Abstract

A method of interfacing a processor bus to a computation engine having a microprogrammable memory-centric controller and an array of memory, comprising the steps of providing a predetermined series of microcode instructions for execution by the MCC; selecting a start address within the series of microcode instructions for carrying out a corresponding operation; and executing the series of microcode instruction in the MCC beginning at the selected start address so as to carry out the corresponding operation in the engine. The present invention is useful in a wide variety of signal processing applications including programmable MPEG encode and decode, graphics, speech processing, image processing, array processors, etc. In telecommunications, the invention can be used, for example, for switching applications in which multiple I/O channels are operated simultaneously.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

INTERNATIONAL SEARCH REPORT

Inte. .ional Application No
PCT/US 98/10549

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/38 G06F9/24 G06F13/16 G11C7/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F G11C

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	ERTEM M C: "A RECONFIGURABLE CO-PROCESSOR FOR MICROPROCESSOR SYSTEMS" PROCEEDINGS OF THE SOUTHEAST CONFERENCE, TAMPA, APRIL 5 - 8, 1987, vol. 1, 5 April 1987, pages 225-228, XP000212298 INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS	1,2,4-7, 16
Y	see the whole document	8-13, 17-20

	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
18 May 1999	26.05.99

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Daskalakis, T

3

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inter. ..ional Application No
PCT/US 98/10549

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
Y	A. C. DAVIES ET AL.: "Interfacing a Hardware Multiplier to a General-purpose Microprocessor" MICROPROCESSORS AND MICROSYSTEMS., vol. 1, no. 7, October 1977, pages 425-431, XP000212024 LONDON GB see page 427	8,11-13
Y	PATENT ABSTRACTS OF JAPAN vol. 097, no. 006, 30 June 1997 & JP 09 034783 A (MITSUBISHI ELECTRIC CORP), 7 February 1997 see abstract	9,10, 17-20
P,Y	& US 5 726 947 A (DOSAKA KATSUMI ET AL) 10 March 1998 see the whole document	9,10, 17-20
X	GB 2 155 671 A (SONY CORP) 25 September 1985 see the whole document	1,2,4-7, 16
X	US 4 862 407 A (FETTE BRUCE A ET AL) 29 August 1989 see the whole document	1,2,4-7
A	US 5 230 042 A (MASAKI YASUO ET AL) 20 July 1993 see the whole document	1,16
A	WO 94 12929 A (S MOS SYSTEMS INC) 9 June 1994 see the whole document	3,4
A	US 5 448 715 A (LELM CHARLES A ET AL) 5 September 1995 see the whole document	9,10, 17-20
A	EP 0 139 254 A (IBM) 2 May 1985 see page 8, line 14 - page 11, line 6	8,11-13
A	DE 44 32 217 A (MITSUBISHI ELECTRIC CORP) 16 March 1995 see column 7, line 52 - column 9, line 50; figure 2	9,10, 17-20
A	EP 0 498 107 A (MITSUBISHI ELECTRIC CORP) 12 August 1992 see the whole document	9,10, 17-20
A	US 5 396 634 A (ZAIDI SYED A A ET AL) 7 March 1995 see column 4, line 1 - column 5, line 11	14,15

3

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 98/10549

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☒ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☒ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. Claims: 1-7, 16

   A computation engine comprising memory, execution unit, controller and bus interface, and a method of interfacing a processor to a computation engine comprising the step of providing a series of microcode instructions in a non-volatile memory accessible to said computation engine by downloading the said series of microcode instructions under processor control.

2. Claims: 8 and 11-15

   A method of interfacing a processor to a memory mapped computation engine

3. Claims: 9, 10 and 17-20

   A computation engine comprising in tandem an SRAM buffer and a DRAM memory array to store data transferred from a processor, and a method of interfacing said processor to said computation engine.

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inte. ational Application No
PCT/US 98/10549

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
GB 2155671 A	25-09-1985	JP 1868550 C	26-08-1994
		JP 58144272 A	27-08-1983
		CA 1193021 A	03-09-1985
		DE 3303488 A	01-09-1983
		FR 2522232 A	26-08-1983
		GB 2115588 A,B	07-09-1983
		NL 8300387 A,B,	16-09-1983
		US 4511966 A	16-04-1985
US 4862407 A	29-08-1989	NONE	
US 5230042 A	20-07-1993	JP 1082272 A	28-03-1989
WO 9412929 A	09-06-1994	JP 8504044 T	30-04-1996
US 5448715 A	05-09-1995	NONE	
EP 0139254 A	02-05-1985	CA 1208802 A	29-07-1986
		JP 1650960 C	30-03-1992
		JP 3016660 B	06-03-1991
		JP 60097458 A	31-05-1985
		US 4814977 A	21-03-1989
DE 4432217 A	16-03-1995	JP 7130166 A	19-05-1995
		US 5521878 A	28-05-1996
		US 5708622 A	13-01-1998
		US 5835448 A	10-11-1998
EP 0498107 A	12-08-1992	JP 4255989 A	10-09-1992
		DE 69126420 D	10-07-1997
		DE 69126420 T	30-10-1997
		KR 9514905 B	16-12-1995
US 5396634 A	07-03-1995	NONE	